Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Denis Božović

Avtomat za zvonjenje na ploščici STM32F4 Discovery

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM
PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: prof. dr. Branko Šter

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Načrtujte in implementirajte avtomat za zvonjenje. Predstavite kratko zgodovino zvonov, načine zvonjenja, različne implementacije zvonjenja in funkcionalnosti, ki jih mora podpirati sodoben avtomat za zvonjenje. Avtomat za zvonjenje implementirajte na ploščici STM32F4 Discovery. Izdelajte tudi uporabniški vmesnik za nastavljanje avtomata.

Želel bi se zahvaliti mentorju prof. dr. Branku Šteru za vodenje pri izdelavi diplomskega dela in strokoven pregled le-tega. Želel bi se tudi zahvaliti družini za vzpodbujanje pri izdelavi diplomskega dela in Dominiku Malovrhu, ki me je vpeljal v obravnavano področje.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
RAM	random access memory	bralno-pisalni pomnilnik
DMA	direct memory access	direktni dostop do pomnilnika
I^2C	inter integrated circuit	
RTC	real time clock	ura realnega časa
OLED	organic light-emitting diode	organska svetleča dioda
SPI	serial peripheral interface	serijski periferni vmesnik
EEPROM	electrically erasable programmable read only memory	električno izbrisljiv programirljiv bralni pomnilnik
SQW	square wave	pravokotni impulz
PWM	pulse-width modulation	pulzno-širinska modulacija

Povzetek

Naslov: Avtomat za zvonjenje na ploščici STM32F4 Discovery

Avtor: Denis Božović

Avtomat za zvonjenje je naprava, katere cilj je v čim večji meri avtomatizirati zvonjenje zvonov in s tem osebo, ki je zadolžena za upravljanje z njimi, razbremeniti te obveznosti. Sodoben način življenja ljudem namreč v veliki meri onemogoči izvajanje zvonjenja, kot je bilo v navadi skozi stoletja. V diplomskem delu je razloženo, kaj se lahko pričakuje od avtomata za zvonjenje zvonov v slovenskem prostoru in zakaj je zaželeno, da podpira določene funkcionalnosti. Na primeru izdelave osnovnega avtomata je pokazan eden izmed načinov, kako lahko tak avtomat v grobem deluje. Za lažje razumevanje so opisane tudi navade zvonjenja v slovenskem prostoru in razvoj avtomatizacije zvonjenja skozi zgodovino.

Ključne besede: zvonjenje, avtomatizacija, mikrokrmilnik, vgrajeni sistemi, STM32F4.

Abstract

Title: Bell automation system on STM32F4 Discovery board

Author: Denis Božović

A bell automation system is a device, the aim of which is to maximize the automation of bell ringing and thus release from duty the person in charge of it. The modern way of life and forms of employment generally make it difficult for human bell-ringers to carry out the task as they did for centuries. In this thesis it is explained what can be expected of the bell automation system in the regions of Slovenia, and why it is desirable that it supports certain functionalities. Using as an example a basic variant of the bell automation system, the fundamental workings of such a device are demonstrated. For easier understanding of some design choices, the basic bell ringing customs in the Slovenian geographic space and the development of bell ringing automation throughout its history, are explained.

Keywords: bell ringing, automatization, microcontroller, embedded systems, STM32F4.

Poglavje 1

Uvod

Cerkveni zvonovi se že več stoletij uporabljajo za naznanjanje časa in bogoslužnih opravil. Najstarejši zvon na Slovenskem je iz 14. stoletja. V Sloveniji imajo cerkve običajno od tri do štiri zvonove, v redkih primerih pa tudi od pet do sedem. Tradicija narekuje, da se mora zvoniti v taktu, kar pomeni, da imajo vsi zvonovi enako hitrost nihanja in udarjajo en za drugim v ritmu. Ob praznikih se je uveljavila tradicija pritrkavanja, ki je skupinski način igranja melodij na zvonove. Zanj so značilne kratke, ponavljajoče se melodije, ki jih običajno izvaja vsaka oseba na svojem zvonu, tako da udarja s kembljem[1] ob zvon, in močan ritem, zaradi česar zahtevajo precejšno usklajenost med člani skupine. Za zvonjenje je bila v preteklosti zadolžena oseba imenovana mežnar, ki je skrbel, da se je redno zvonilo. Občasno so mu priskočili na pomoč tudi sovaščani, saj so zvonovi zelo težka glasbila. Tak način zvonjenja se je nadaljeval vse do 20. stoletja, ko se je zvonjenje počasi začelo avtomatizirati [11, str. 15, 22].

Avtomat za zvonjenje je naprava, ki omogoča delno ali popolno avtomatizacijo zvonjenja cerkvenih zvonov. Običajno opravlja sledeče naloge:

- hranjenje vnosov časa in načina zvonjenja ter izvršitev zvonjenja ob pravem času,

[1] Tolkač znotraj zvona, ki ob nihanju udarja ob zvon in s tem povzroči vibriranje zvona, kar ustvarja zvok.

- bitje ure na zvonove po vnešenem vzorcu,
- premikanje urinih kazalcev in
- v nekaterih primerih upravljanje z zunanjo razsvetljavo.

Razlika med zmogljivejšimi in manj zmogljivimi avtomati je predvsem v tem, koliko različnih možnosti dopuščajo pri programiranju zvonjenja, na kakšen način upravljajo z zvonovi in koliko dodatkov, ki olajšajo pogosta opravila (npr. daljinsko krmiljenje med pogrebi) imajo.

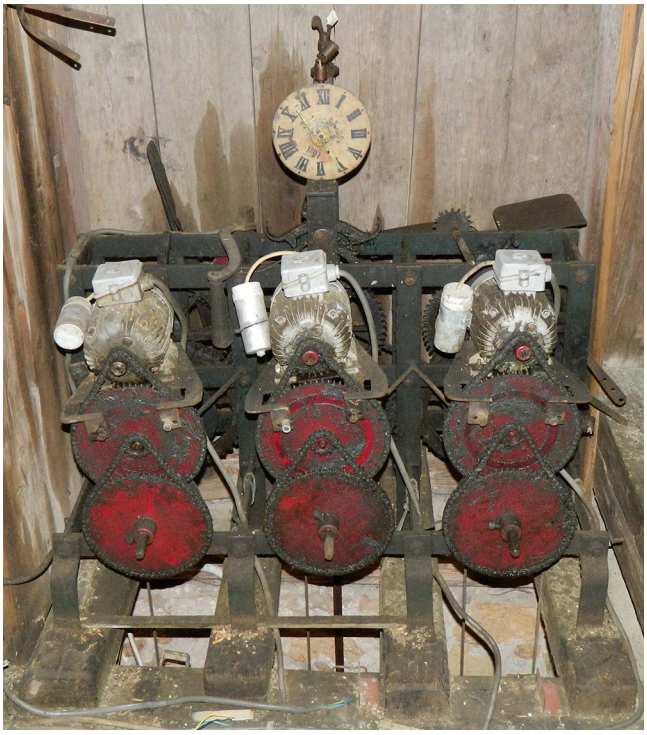
V diplomskem delu smo razložili pojme in navade povezane z zvonjenjem cerkvenih zvonov in kako ter kje te pridejo v poštev pri avtomatizaciji. Izdelali smo tudi avtomat za zvonjenje, ki je zmožen krmiliti majhne zvonove.

Poglavje 2

Zgodovina avtomatizacije zvonjenja

Prva resnejša oblika avtomatizacije so bile mehanske ure (na sliki 2.1), ki so se na Slovenskem zelo razširile v 19. stoletju. Te so običajno vsak četrt ure silo gravitacije preko uteži in vzvodov spreminjale v udarce kladiv za bitje ure in premike kazalcev, niso pa mogle zvoniti. Še vedno jih je bilo potrebno navijati na nekaj dni, zaradi velikih temperaturnih razlik med letnimi časi pa se je tudi natančnost precej spreminjala, saj se je nihalo ob mrazu skrajšalo, ob vročini pa podaljšalo in s tem spremenilo svoj nihajni čas. Uro je bilo zato potrebno budno spremljati in nihajni čas ročno popravljati z vrtenjem matice na dnu nihala. To delo je običajno opravljal mežnar.

Čeprav se je v tujini zvonjenje začelo avtomatizirati že na začetku 20. stoletja, se je v slovenskih krajih zaradi posebnih zahtev, ki jih ima zvonjenje v taktu, avtomatiziranje začelo razširjati šele v drugi polovici stoletja. Prvi poskusi so bili elektro-mehanski, najboljši analogiji za predstavo pa sta pogonska gred parne lokomotive ali cilindri današnjega avtomobilskega motorja (na sliki 2.2). Hitrost vrtenja gredi in nihajni čas zvonov sta seveda morala biti skladna. Tak pogon je imel nekaj hudih pomanjkljivosti. Zvon je bil trdo mehansko vezan na svoj pogon. Če se je zgodil izpad električne energije tekom zvonjenja, so zvonovi obstali v nenaravnem položaju in oprema



Slika 2.1: Popolnoma mehanska ura s kasneje dodanim avtomatskim navijanjem

se je zato lahko poškodovala. Iz istega razloga pa je bilo nihanje zvonov "prisiljeno". Prav tako ni bilo mogoče ročno zvoniti ob izjemnih dogodkih, kot so obujanje tradicije ročnega zvonjenja ali izpad električne energije, saj trda mehanska povezava ni omogočala prostega gibanja. Del avtomatizacije, ki je zvonjenje sprožal, je bil sprva stenska mehanska ura na nihalo z dodanimi stikali na mehanizmu. Ko je bil čas za zvonjenje ali bitje ure, je zatič na kolesju sklenil stikalo in tako sprožil zvonjenje ali bitje. Marsikje so za potrebe bitja ure ohranili stare ure iz 19. stoletja in jim dodali avtomatsko navijanje.

Z izumom pogona na verigo (na sliki 2.3), ki se je razvil v več različic, je bila marsikatera slabost preteklega pogona odpravljena, saj je zvon lahko prosto nihal, medtem ko je bil pogon izključen, hkrati pa ni bil več trdo mehansko vezan nanj, kajti veriga predstavlja mehak člen. Ključen napredek

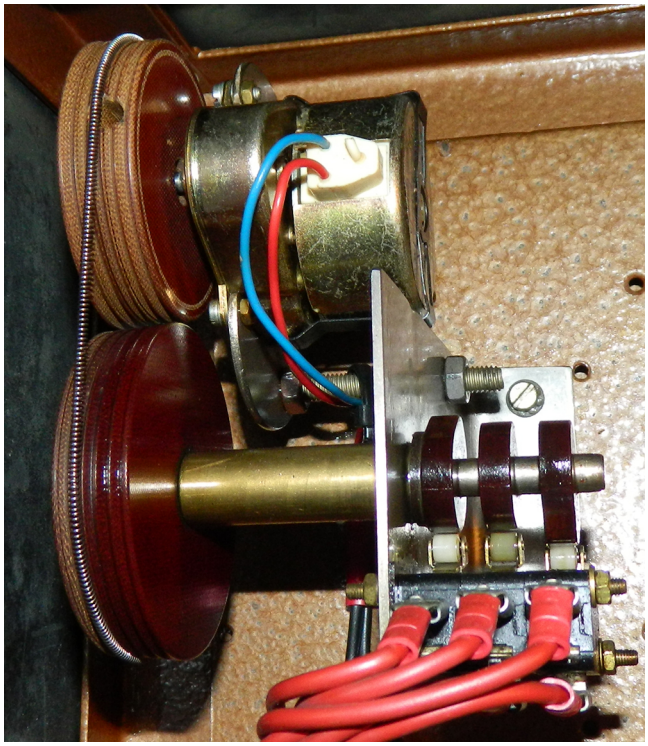
Slika 2.2: Pogon z vrtečo se gredjo in pogonskimi palicami, pritrjenimi neposredno na zvonove

je predstavljala zamisel, da se lahko pogonska gred, omenjena pri prejšnjem pogonu, preoblikuje in pomanjša ter prestavi v elektro omaro (na sliki 2.4). Gred ni bila več mehansko vezana na zvonove, temveč je med vrtenjem vklapljala in izklapljala stikala ter tako krmilila elektromotorje, ki so bili preko verige povezani z zvonom (analogija z glasbenimi omarami, kjer so zatiči na vrteči se gredi zavibrirali posamezne glasbene palice). Sprva so obstajale le različice, ki so delovale enosmerno (motor je bil aktiviran le za vleko v eno smer, v drugo pa je zvon prosto zanihal), kasneje pa so se pojavile tudi dvosmerne, ki so omogočale boljši ritem. Ker na tak način ni mogoče aktivno upravljati s hitrostjo in močjo elektromotorja, je bilo potrebno dodati med elektromotor in zvon dodaten zobnik, ki je zmanjšal hitrost motorja in tako

preprečil, da bi zvon nihal previsoko. Izbira premočnega motorja je predstavljala problem v preveč nasilnih zagonih, izbira prešibkega pa je povzročila dolg čas zahajanja v želen takt. Takšne pogone še danes pogosto najdemo v Sloveniji. Njihova glavna slabost je, da so pogosto preveč nasilni in tako predvsem pri zagonu trpita tako oprema kot tudi zvon, težko pa je tudi optimizirati gibanje za najboljši možen zven, takt in različne tipe kembljev.

Slika 2.3: Ena izmed starejših različic pogona na verigo

Avtomat za sprožanje zvonjenja je sčasoma izgubil nihalo, nadomestila ga je elektronika, ki je oddajala impulze vsako minuto. Za vnos zvonjenj se je pojavilo veliko kolo, ki ga je vrtel mehanizem ure. S privijanjem vijakov v luknje v kolesu se je določil urnik zvonjenja. Pod kolesom so bila nameščena stikala, tako da so med obračanjem kolesa vijaki pritiskali na stikala in s tem aktivirali različne programe zvonjenja (na sliki 2.5). Kompleksnejši programi zvonjenja so bili prav tako realizirani z vrtečo se gredjo

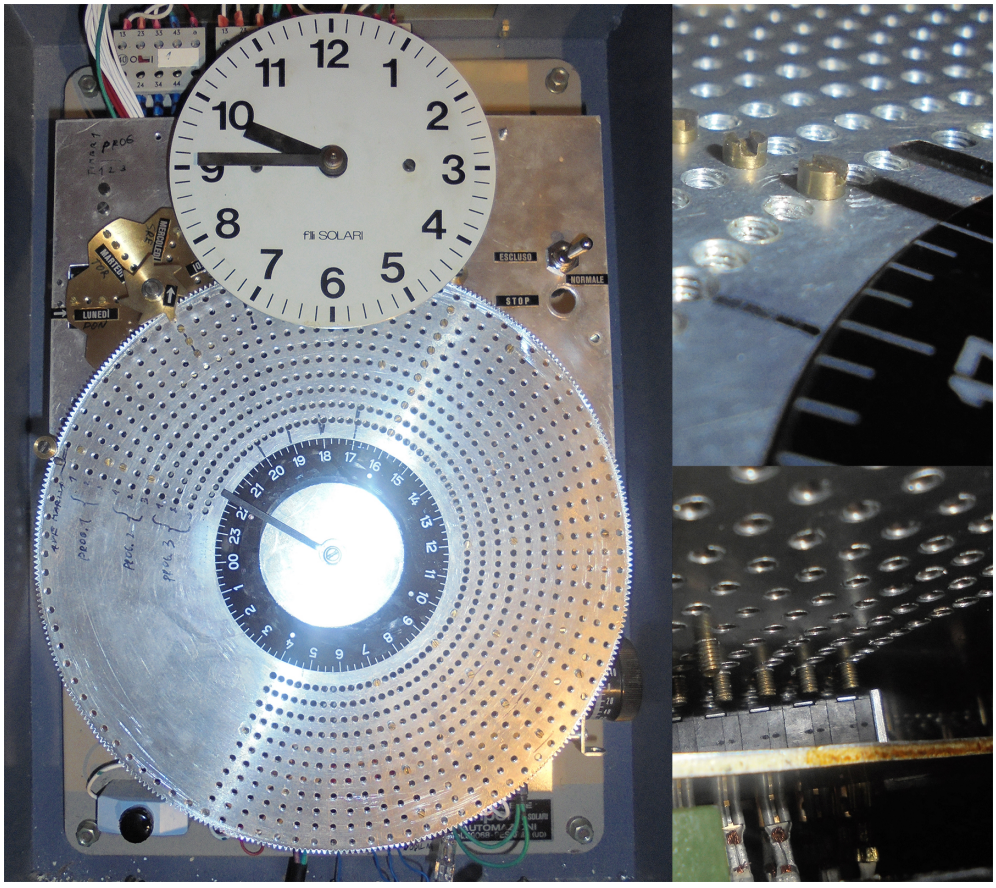


Slika 2.4: Vrteča se gred z zatiči in stikali

z zatiči, ki so ob vrtenju pritiskali na stikala. Ker je bila ta rešitev še vedno elektro-mehanska, je omogočala le statične tedenske vnose zvonjenj, mrliško zvonjenje ter zelo omejeno število melodij in programov, saj bi v nasprotem primeru kompleksnost mehanike in s tem cena strmo narasli.

Z razvojem računalništva so se mnogi deli avtomatizacije lahko pomanjšali in predali v skrb računalniku, razširila pa se je lahko tudi funkcionalnost. Tako je računalnik sčasoma prevzel funkcijo skrbnika nad taktom, olajšal vnašanje zvonjenj in razširil možnosti programiranja zvonjenja, ki so pri tem na voljo, omogočil daljinsko krmiljenje, enostavne popravke takta in še bi lahko naštevali. Ker je za izvedbo pritrkovalskih melodij običajno potrebno zbrati večjo skupino ljudi, kar je ob današnjem tempu življenja težje izvedljivo kot včasih, so avtomati začeli podpirati tudi to funkcijo.

Moderna rešitev za avtomatizacijo (primer na sliki 2.6) uporablja računalniško vodeno frekvenčno regulacijo [19], ki omogoča krmiljenje motorja



Slika 2.5: Elektromehanski avtomat za zvonjenje in bitje ure

na način, ki v čim manjši meri izrablja zvon in njegovo opremo, omogoča pa tudi fino nastavljivost nihanja in preprosto realizacijo zavor, kar je v prejšnjih avtomatizacijah precej težje izvedljivo. V primeru obnove starejših pogonov je mogoče dodati vezje, ki omili nasilne potege motorjev in tako reši nekaj hujših problemov starejših avtomatizacij.

Slika 2.6: Sodoben avtomat za zvonjenje

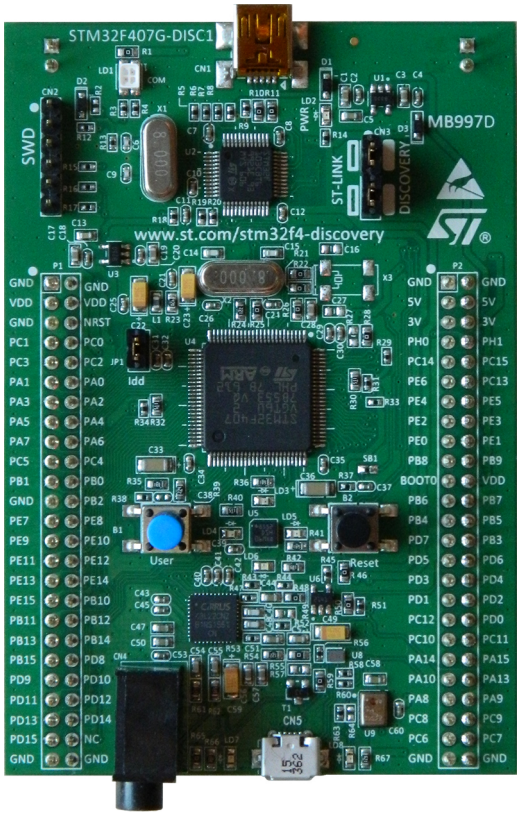
Poglavje 3

Strojna oprema

3.1 Mikrokrmilnik

Jedro sistema predstavlja ploščica STM32F4 Discovery (na sliki 3.1) podjetja ST Microelectronics, ki temelji na visoko zmogljivem mikrokrmilniku STM32F407VG [18]. Izbrana je bila, ker je cenovno ugodna, a vseeno zmogljiva rešitev. Pomembnejši deli omenjenega mikrokrmilnika, ki se jih uporablja v tem diplomskem delu, vključujejo:

- 32-bitni procesor Cortex-M4 podjetja ARM z enoto za računanje v plavajoči vejici, ki deluje s frekvenco do 168MHz,
- 1MB pomnilnika flash,
- 192 kB RAMa,
- splošno uporaben DMA krmilnik,
- 17 različno zmogljivih časovnikov,
- 140 vhodno-izhodnih priključkov in
- 3 I^2C vmesnike.



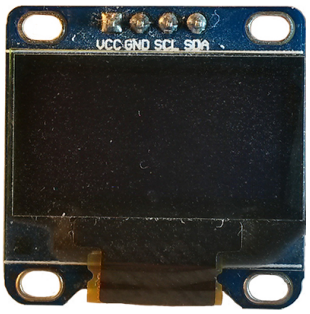
Slika 3.1: Ploščica STM32F4 Discovery

Za programiranje mikrokrmilnika smo izbrali programski jezik C++, ker smo se poslužili nekaterih naprednih konceptov, ki jih uvaja, kot so npr. razredi (*classes*), vzorci (*templates*) in dedovanje [10][5]. Za prevajanje, razhroščevanje in prenos programa na STM32F4 Discovery smo uporabili orodje IAR Embedded Workbench [22], za urejanje in pisanje pa orodje Microsoft Visual Studio 2013 [6].

3.2 Prikazovalnik

Za prikaz grafike smo izbrali manjši monokromatski OLED zaslon kitajskega porekla (na sliki 3.2), ki je kopija enakega izdelka podjetja Adafruit. Ima diagonalo zaslona 0,96" (2.4384 cm) in ločljivost $128 * 64$ slikovnih točk,

upravlja pa ga gonilnik z oznako SSD1306 [21]. Razlogi za izbiro so bili nizka cena, v javnosti pogosto uporabljan gonilnik, zadovoljiva ločljivost ter podpora I^2C protokolu (do 400 kHz), ki je ena izmed najbolj pogostih izbir med protokoli za komunikacijo med vgrajenimi sistemi na kratke razdalje [20].



Slika 3.2: Zaslon

S prikazovalnikom upravlja knjižnica TM SSD1306 avtorja Tilna Majerleta [15], ki zaslon inicializira in mu na zahtevo pošilje izrisano sliko, ki se nahaja v pomnilniku mikrokrmilnika. Vsebuje tudi funkcije za izris slike, s katerimi lahko izrisujemo osnovne like, kot so trikotnik, štirikotnik, črta, krog in točka, izpisujemo besedilo ter nad sliko vršimo nekatere učinke, kot je invertiranje slike.

Za komunikacijo uporablja knjižnico TM I2C istega avtorja [14], ki poenostavi uporabo I^2C protokola. Omogoča, da v enem klicu funkcije izvršimo pošiljanje ali prejemanje podatkov, hkrati pa lahko pred pošiljanjem samih podatkov pošljemo tudi naslov, kar se običajno uporablja za navajanje registra na napravi, iz katerega želimo brati oz. v katerega želimo pisati. Knjižnico TM I2C smo morali nekoliko spemeniti, saj je privzeto uporabljala vgrajene upore mikrokrmilnika, ki pa so bili previsoke upornosti za dovolj hitro stabilizacijo linije.

Knjižnico TM SSD1306 smo nadgradili z novim, fleksibilnejšim načinom shranjevanja pisav, ki je kompatibilen s pisavami, generiranimi v programu za generiranje pisav The Dot Factory [7]. Dodali smo tudi podporo DMA prenosom, kar zelo razbremeni mikrokrmilnik, saj je za vsako osvežitev za-

slona potrebno prenesti 1024 B podatkov. Slednjo nadgradnjo je za ploščice družine STM32F103xx implementiral uporabnik spletne strani avtorja Sir-Volta [24], mi pa smo jo prepisali za uporabo s ploščico STM32F4 Discovery in dodali prekinitve ob končanju I^2C prenosa. Ta nadgradnja se je dotaknila tudi knjižnice TM I2C.

Za lažjo zamenjavo zaslona in knjižnice smo vse klice, ki se nanašajo na izris grafike, preusmerili preko vmesnega razreda *GfxMediator*. Tako je mogoče v primeru, ko bi bilo potrebno knjižnico zamenjati, enostavneje spremeniti program, saj se spremeni le razred *GfxMediator*.

3.3 Modul za uro realnega časa

Čeprav ima mikrokrmilnik STM32F407VG vgrajeno uro realnega časa, smo raje izbrali zunanjo rešitev ChronoDot podjetja Macetech (na sliki 3.3) [13], ki je natančnejša. Vključuje temperaturni senzor, ki hitrost oscilatorja prilagaja glede na temperaturo okolice, možno pa je tudi popravljati natančnost, ki se izgubi s staranjem kristala. Podpira tudi generiranje pravokotnih impulzov, na izbiro so 4 možne frekvence. Podatke pošilja preko protokola I^2C s hitrostjo do 400 kHz.

Slika 3.3: Modul za uro realnega časa ChronoDot

Za komunikacijo z modulom smo izbrali knjižnico, ki so jo spisali pri JeeLabs, za platformo Arduino pa prilagodila Stephanie Maks [16]. Knjižnico smo morali prilagoditi za mikrokrmilnik STM32F407VG. Največja sprememba

je bila zamenjava ukazov knjižnice Wire (ki skrbi za I^2C komunikacijo na Arduino platformi) z ukazi knjižnice TM I2C, potrebnih pa je bilo še nekaj manjših popravkov in dodatkov, kot je npr. možnost generiranja pravokotnih impulzov z eno izmed 4 možnih frekvenc na SQW izhodu modula. Pomembnejše funkcionalnosti knjižnice vključujejo:

- branje podatkov o trenutni uri in datumu,
- nastavljanje trenutne ure in datuma,
- implementacija preprostega razreda za hranjenje datuma in
- nastavljanje generiranja pravokotnih impulzov z eno izmed 4 frekvenc, ki so na izbiro, na SQW izhodu.

Da bi komponente avtomata, ki uporabljajo uro realnega časa, vedno vedele, ali je ura v nekem trenutku posodobljena in da bi bilo osveževanje ure čim manj blokirajoče, se uporabljajo naslednja stanja ure:

- ura posodobljena,
- ura pravkar posodobljena in
- ura ni osvežena.

Kadar je stanje ure *ura posodobljena*, je ura v avtomatu sinhronizirana z uro v modulu realnega časa in jo je mogoče varno uporabiti. Stanje *ura pravkar posodobljena* je uporabljeno za obveščanje komponent, ki izvršujejo funkcije enkrat na sekundo, da se je ura pravkar spremenila, in se takoj po obveščanju spremeni v stanje *ura posodobljena*. Ko modul realnega časa preko prekinitve mikrokrmilnik obvesti, da se je ura spremenila, se stanje prestavi v *ura ni osvežena*, dokler se aktualen čas ne prenese iz modula realnega časa in se nastavi stanje *ura pravkar posodobljena*.

3.4 Trajno hranjenje podatkov

Za trajno hranjenje podatkov smo izbrali zunanji EEPROM z oznako 24FC512-I/P proizvajalca Microchip Technology (na sliki 3.4) [17]. Z velikostjo 512 kb pusti dovolj prostora za morebitne kasnejše razširitve funkcionalnosti. Komunikacija poteka preko I^2C protokola s hitrostjo do 1 MHz. Omogoča tudi zaporedno neprekinjeno branje celotnega pomnilniškega prostora, zaporedno neprekinjeno pisanje po straneh do 128 b in izbiro enega izmed 8 I^2C naslovov, ki so na voljo.

Za uporabo EEPROMa smo napisali paket funkcij, ki lahko kateri koli podatkovni tip, pretvorjen v seznam bajtov, pišejo in berejo iz EEPROMa, na voljo pa je tudi funkcija za prepis EEPROMa z vzorcem 0xFF. Za varčevanje s številom pisanj v EEPROM se pred pisanjem pregleda, ali so obstoječi podatki v EEPROMu enaki tistim, ki jih želimo zapisati in v kolikor so, se pisanje ne izvede. Za samo pošiljanje podatkov smo uporabili knjižnico TM I2C. Ker uporabljeni EEPROM potrebuje 16 bitov za izbiro naslova, smo morali knjižnico TM I2C razširiti s funkcijami, ki omogočajo pošiljanje 16-bitnega naslova.

Ob zagonu se na začetku naslovnega prostora EEPROMa preveri, ali je vpisan podpis avtomata, ki zagotavlja, da so podatki v EEPROMu veljavni [2]. Če tega podpisa ni, se vsi podatki iz EEPROMa pobrišejo, da ne bi prišlo do napačnega delovanja. Za tem se glede na velikosti parametrov, ki jih v EEPROM shranjujejo različni sklopi avtomata, izračunajo in dodelijo naslovi, kamor posamezen sklop sme shranjevati svoje parametre. Sklopi si zaporedoma rezervirajo potreben del prostora v EEPROMu, velikost tega prostora pa se po vsaki rezervaciji prišteje števcu naslovov. Tako se prepreči, da bi se rezervirani naslovni prostori različnih sklopov prekrivali. Podoben pristop so uporabili v knjižnici [8]. Na koncu se vpiše podpis avtomata v obliki treh črk in treh števil, ki predstavljajo različico programske opreme.

Slika 3.4: EEPROM 24FC512-I/P

Poglavje 4

Uporabniški vmesnik

Uporabniški vmesnik uporabniku omogoča vpogled v trenutno stanje avtomata in njegovo nastavljanje. V grobem je sestavljen iz menijskega sistema in vhodnega sistema. Celoten vmesnik je načrtovan s poudarkom na izogibanju dinamični alokaciji pomnilnika.

4.1 Vhodni sistem

Vhodni del uporabniškega vmesnika je prva stopnja uporabnikove interakcije z avtomatom. Njegova naloga je zajem uporabnikovih ukazov (pritiskov tipk, premikov stikal) in posredovanje le-teh menijskemu sistemu, ki predstavlja drugo stopnjo uporabnikove interakcije. Sestavljajo ga tipke za pomikanje po menijskem sistemu in stikala, ki omogočajo takojšnjo uporabo najpogosteje rabljenih funkcij avtomata. Posamezne tipke so predstavljene kot instance razreda *HW_Key*, katerega deklariramo s proceduro 4.1. Kot parametre sprejema:

- fizični priključek, na katerega je priključena tipka,
- eno izmed vnaprej definiranih akcij, ki jih lahko opravlja tipka (naprej, nazaj, izberi, gor, dol in izhod),
- stabilizacijski čas tipke,

- privzeto stanje tipke, ki je lahko visoko ali nizko ter
- nastavitev za vklop ali izklop držanja tipke, ki na določen interval ponavlja ukaz tipke ob držanju.

```
1    HW_Key(
2         uint32_t ahbPeriph,
3         GPIO_TypeDef* gpio,
4         uint16_t pin,
5         KeyRole key_role,
6         uint16_t debounce_delay,
7         KeyLogicLevel default_key_state,
8         bool enable_hold = false
9         );
```

Procedura 4.1: Definicija tipke

Razredu po deklaraciji vseh tipk podamo seznam le-teh. Vsako milisekundo časovnik sproži prekinitev, ki sproži postopek branja stanja vseh tipk v seznamu. Pritisk tipke se zazna kot veljaven šele, ko poteče podan stabilizacijski čas, saj ima večina tipk zaradi svojih mehanskih lastnosti določen čas umirjanja signala [9]. Če se po poteku stabilizacijskega časa stanje tipke ne spremeni, se to zazna kot uporabnikov pritisk in akcija, katero tipka predstavlja, se doda v krožni seznam. Tam ostane, dokler je menijski sistem ne prevzame oz. dokler se krožni seznam ne zapolni in je ena izmed naslednjih akcij ne prepiše.

Za uporabo avtomata smo predvideli 6 tipk:

- tipki **naprej** in **nazaj** primarno služita pomikanju naprej in nazaj po komponentah ali vrsticah na zaslonu,
- tipki **gor** in **dol** sta namenjeni interakciji z izbrano komponento, običajno je to povečevanje in zmanjševanje številskih vrednosti,
- tipka **izberi** je uporabljena za potrditev vnešenih podatkov in pomik na naslednji meni ter

- tipka **izhod** je namenjena premiku za en meni nazaj.

Pri izbiri tipk smo pazili, da avtomat ne bi bil težak za uporabo zaradi premajhnega števila tipk. Število 6 je tako predstavljalo najmanjšo izbiro, s katero je avtomat še mogoče preprosto upravljati.

4.2 Menijski sistem

Menijski sistem uporabniku omogoča nastavljanje avtomata, tako da interpretira njegove ukaze in pripravi ustrezen odziv nanje, bodisi preko knjižnice za izris grafike ali z aktivacijo funkcije avtomata. Sestavljajo ga vsebnik in posamezne komponente. Pri snovanju izgleda uporabniškega vmesnika smo se zgledovali po [12].

4.2.1 Vsebnik

Vsebnik je podlaga vsakega menija. Hrani seznam vseh komponent, ki so del menija, jim posreduje uporabnikove ukaze ter kliče njihove funkcije za izris. Vsak vsebnik hrani trenutno stanje svojega menija (tj. katera komponenta je trenutno izbrana), ki ga je preko klica funkcije mogoče ponastaviti na začetne vrednosti. Omogoča tudi nastavljanje poljubnih povratnih funkcij za prestrezanje pritiskov tipk, inicializacijo ter ponastavitev stanja. Statična spremenljivka razreda hrani trenutno izbran (aktiven) vsebnik. Vsebnik se uporabi s proceduro 4.2,

```
1 UI_Container(
2     UI_Item** element_list, uint8_t num_of_el,
3     bool(*keyCallbackFunc)(KeyCommand) = 0,
4     void(*containerInitCallbackFunc)(void) = 0,
5     void(*resetCallbackFunc)(void) = 0
6     );
```

Procedura 4.2: Definicija vsebnika

kjer se kot parametre po vrsti navede seznam komponent, dolžino seznama, nato pa tri opcijske povratne funkcije:

- funkcijo za prestrezanje pritiskov tipk, ki omogoča, da prestrežemo katerikoli ukaz tipke, namenjen trenutno aktivnemu vsebniku,
- funkcijo za inicializacijo, ki jo uporabimo, kadar želimo, da se neka procedura izvede le ob zagonu in
- funkcijo za ponastavitev stanja komponent.

Po potrebi je mogoče določiti tudi povratno funkcijo, ki se izvede vsakokrat, ko se zamenja trenutno aktiven vsebnik.

Da bi menijski sistem povzročil čim manj nepotrebnega osveževanja zaslona in da bi bil postopek neblokirajoč, se uporabljajo naslednja stanja zaslona:

- zaslon osvežen,
- potreben izris slike in
- potreben prenos slike.

Ko je aktivno stanje *zaslon osvežen*, je trenutno izrisana slika na zaslonu aktualna in potrebna ni nobena sprememba. Vsakič, ko uporabnik pritisne tipko in se le-ta posreduje trenutno aktivnemu vsebniku, se stanje spremeni v *potreben izris slike*, saj uporabnikove akcije v skoraj vseh primerih povzročijo spremembo stanja menijev. To stanje povzroči klic funkcije za ponovni izris aktivnega vsebnika (ki nato izriše vse vsebovane komponente). Po izrisu se stanje spremeni v *potreben prenos slike*, ko se slika iz pomnilnika mikrokrmilnika prenese na zaslon. To zaključi postopek osvežitve zaslona, zato se stanje spremeni nazaj v *zaslon osvežen*.

4.2.2 Izbira

Izbira (fokus) je nujen del vsakega menijskega sistema, saj brez le-te uporabnik ne more izbirati, čemu želi namenjati ukaze tipk.

Vsaka komponenta menijskega sistema, pri kateri je smiselno omogočiti interakcijo, ima implementirane odzive na pritiske tipk. Tako ima npr. drsni

seznam vgrajeno možnost pomikanja po seznamu in odzive na vsak možen izbor. Vsebnik ob začetku poišče prvo komponento, ki podpira možnost interakcije in ji sporoči, da je izbrana. To naredi tako, da komponentam po vrsti ponudi izbiro, te pa lahko izbiro sprejmejo ali zavrnejo. Naenkrat je lahko izbrana le ena komponenta vsebnika. Komponente nato vsakič, ko jim vsebnik preda akcijo tipke, odgovorijo s sporočili o posledicah uporabnikovih akcij, ki so lahko:

- ohranitev izbire pri trenutni komponenti,
- predaja izbire naslednji komponenti ali
- predaja izbire prejšnji komponenti vsebnika.

Vsebnik glede na odgovor bodisi ne naredi ničesar ali sporoči naslednji oz. prejšnji komponenti na seznamu (odvisno od odgovora), da je sedaj izbrana.

4.2.3 Komponente

Komponente so gradniki menijskega sistema, preko katerih le-ta dobi funkcionalnost in izgled. Zasnovane so kot objekti, ki dedujejo iz razreda *UI_Item* [1], kar omogoča, da so vse v istem seznamu ter je na njih moč klicati virtualne funkcije, ki so skupne vsem komponentam. Le-te vključujejo:

- funkcijo za izris komponente,
- funkcijo za izvrševanje ukazov tipk,
- funkcijo za ponastavitev stanja,
- funkcijo za dodeljevanje izbora (fokusa),
- funkcije za nastavljanje in pridobivanje lokacije,
- funkcijo za izklop komponente in
- funkcije za nastavljanje in pridobivanje dimenzij.

Drsni seznam

Drsni seznam je komponenta, ki omogoča vertikalno pomikanje po seznamu naslovov in njihovo izbiro. Uporabljen je predvsem za pomikanje med meniji (primer na sliki 4.1). Implementiran je kot razred *UI_ScrollMenu*, ki v sebi hrani:

- seznam naslovov, ki se izpišejo po vrsticah in njegovo dolžino,
- podatke o izgledu seznama, ki vključujejo število vidnih vrstic na zaslonu, višino posamezne vrstice, barvo in obliko pisave,
- podatke o trenutnem stanju seznama, ki vključujejo trenutno izbrano vrstico ter zgornjo in spodnjo na zaslonu vidno vrstico ter
- naslov funkcije, ki izvrši akcijo ob izboru nekega naslova (običajno je to premik na izbran meni).

Po seznamu se pomikamo s tipkama *naprej* in *nazaj*, naslov pa izberemo s tipko *izberi*. Uporabi se ga s proceduro 4.3.

```
1  UI_ScrollMenu(
2      uint16_t x,
3      uint16_t y,
4      uint8_t line_height,
5      uint8_t visible_lines,
6      uint16_t menu_width,
7      GM_Font font,
8      GM_Color color,
9      uint8_t num_of_items,
10     char** line_titles,
11     void(*selectCallbackFunc)( uint8_t)
12     );
```

Procedura 4.3: Definicija drsnega seznama

Splošen drsni seznam

Splošen drsni seznam omogoča, da v vertikalnem seznamu uporabimo skoraj katero koli komponento menijskega sistema, ki smiselno spada v ta kontekst in podpira možnost izbire (primer na sliki 4.4). Implementiran je v razredu *UI_ScrollList*. Namenjen je predvsem grupiranju komponent v seznam, ki lahko sega preko roba zaslona. V primeru, ko se število komponent v meniju spremeni, to precej olajša prilagoditev menija. Pomikanje po takem seznamu prav tako vršimo s tipkama *naprej* in *nazaj*, interakcijo s trenutno izbrano komponento v vrstici pa vršimo s tipkami, ki se tudi običajno uporabljajo z uporabljeno komponento. Splošen drsni seznam se uporabi s proceduro 4.4. Na voljo je tudi horizontalna različica, implementirana v razredu *UI_HScrollList* (primer na sliki 4.3).

```
1  UI_ScrollList(
2      uint16_t x,
3      uint16_t y,
4      uint8_t line_height,
5      uint8_t visible_lines,
6      uint16_t menu_width,
7      uint8_t num_of_lists,
8      uint8_t* lists_lengths,
9      UI_Item*** line_lists,
10     void(*confirmCallbackFunc)()
11     );
```

Procedura 4.4: Definicija splošnega drsnega seznama

Oznaka

Oznaka omogoča postavitev statičnega besedila v meni (primer na sliki 4.1). Implementirana je v razredu *UI_Label*, ki vsebuje kazalec na besedilo, ki naj bo prikazano, dimenzije oznake in barvo ter slog pisave. Uporabi se jo s proceduro 4.5.

```
1 UI_Label(
2     uint16_t x,
3     uint16_t y,
4     uint16_t height,
5     uint16_t width,
6     GM_Font font,
7     GM_Color color,
8     char* text
9     );
```

Procedura 4.5: Definicija oznake

Celo število

Komponenta za vnos celega števila (primer na sliki 4.3) je ena izmed nepogrešljivih, saj uporabniku omogoča spreminjanje in vnos številskih nastavitev v avtomat. Implementirana je kot razred *UI_IntNumber*, ki vsebuje:

- trenutno izbrano število,
- interval, na katerem sme biti izbrano število,
- največjo možno širino prikazanega števila,
- slog pisave in barvo komponente,
- podrobnosti o načinu prikaza števila (prikaži plus, vodilne ničle itn.),
- nastavitev, ali naj se število prelije, ko doseže konec intervala in
- naslov funkcije, ki se kliče vsakič, ko uporabnik spremeni število, če se to potrebuje (npr. pri datumih).

Razred je načrtovan kot predloga (*template*) in tako omogoča uporabo s poljubnim celoštevilskim tipom spremenljivke [23]. Če je prikazano število ožje od najširšega možnega števila, se poravna desno. Število je mogoče spreminjati s tipkama *gor* in *dol*. Komponento se uporabi s proceduro 4.6.

```
UI_IntNumber(
    uint16_t x,
    uint16_t y,
    T min_number,
    T number,
    T max_number,
    GM_Font font,
    GM_Color color,
    char* additional_parameters = "",
    bool(*changeCallbackFunc)(T) = 0
    );
```

Procedura 4.6: Definicija komponente za izbiro celega števila

Krožna izbira

Komponenta za krožno izbiro (primer na sliki 4.2) je splošnejša oblika komponente za izbor celega števila. Vsakemu elementu v zalogi vrednosti je dodeljen indeks, prikažemo pa ga z uporabo poljubne funkcije za izris. Tako lahko za predstavitev elementa na zaslonu uporabimo od poljubnega niza znakov do poljubne grafike. Po potrebi lahko navedemo tudi povratno funkcijo, ki se bo klicala ob vsaki spremembi izbire. Komponenta je implementirana v razredu *UI_Spinner*, ki vsebuje:

- trenutno izbrani indeks,
- največji dovoljeni indeks,
- barvo komponente,
- višino in širino komponente ter
- naslov funkcije za izris.

Komponenta je uporabljena predvsem v primerih, ko se nekaj izmed elementov na izbiro drastično razlikuje od ostalih, npr. med številskimi vrednostmi želimo tudi izbiro, ki pomeni “ni izbrano” in se označuje z znakom “-”. Uporabi se jo s proceduro 4.7.

```
UI_Spinner(
    uint16_t x,
    uint16_t y,
    uint16_t h,
    uint16_t w,
    uint8_t num_el,
    GM_Color color,
    void(*renderCallbackFunc)(uint8_t i, uint16_t x,
        uint16_t y, uint16_t h, uint16_t w),
    bool(*changeCallbackFunc)(uint8_t i) = 0
    );
```

Procedura 4.7: Definicija komponente za krožno izbiro

Drsni trak

Drsni trak je komponenta, ki uporabniku olajša pomikanje po seznamih s tem, da mu daje povratno informacijo o trenutno izbrani lokaciji v seznamu glede na velikost celotnega seznama v obliki drsnega draku (primer na sliki 4.1). Ker uporablja predloge (*templates*), jo je mogoče uporabiti s katero koli komponento, ki vsebuje funkcijo za pridobivanje indeksa trenutno izbranega elementa (kot je npr. drsni seznam). Implementirana je v razredu *UI_Vsb*, ki vsebuje kazalec na povezano komponento, dimenzije in barvo traku. Drsni trak se uporabi s proceduro 4.8.

```
UI_Vsb(
    T* linked_item,
    uint16_t x,
    uint16_t y,
    uint16_t height,
    uint16_t width,
    GM_Color color
    );
```

Procedura 4.8: Definicija drsnega traku

Poljubno

Komponenta poljubno omogoča integracijo poljubne funkcije za izris v obstoječ menijski sistem (primer na sliki 4.14). Implementirana je v razredu *UI_Custom*, ki hrani kazalec na funkcijo za izris. Uporabimo jo s proceduro 4.9.

```
1 UI_Custom(
2     uint16_t x,
3     uint16_t y,
4     void(*renderCallbackFunc)( uint16_t,  uint16_t)
5 );
```

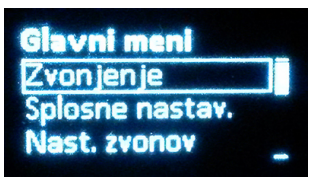
Procedura 4.9: Definicija komponente poljubno

4.3 Uporaba menijev

Sledi opis uporabe nekaterih menijev, ki omogočajo osnovno delovanje avtomata.

4.3.1 Navigacija

Za pomikanje med končnimi meniji se uporablja mreža med seboj povezanih navigacijskih menijev, ki se začne v glavnem meniju in z naslovi vodi uporabnika do ciljnega končnega menija (primer na sliki 4.1). Po naslovih podmenijev se pomikamo s tipkama *naprej* in *nazaj*, podmeni pa izberemo s tipko *izberi*.



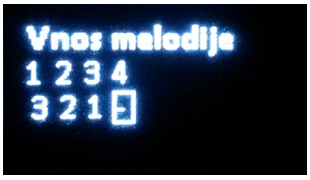
Slika 4.1: Primer navigacijskega menija

Za realizacijo navigacijskih menijev se uporabljajo komponente:

- oznaka za prikaz naslova menija,
- drsni seznam za prikaz in pomikanje med naslovi podmenijev ter
- drsni trak za lažjo orientacijo v seznamu naslovov.

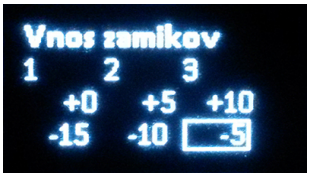
4.3.2 Vnos vzorca zvonjenja

Vzorec zvonjenja se vnaša v dveh delih. V prvem delu vnesemo melodijo zvonjenja (na sliki 4.2). S tipkama *naprej* in *nazaj* se pomikamo med polji, ki predstavljajo zvonove, s tipkama *gor* in *dol* pa izberemo mesto, ki ga bo izbran zvon zasedel v melodiji. Če zvona ne želimo uporabljati, pustimo izbrano vodoravno črto.



Slika 4.2: Meni za vnos melodije

Po pritisku tipke *izberi* se prikaže drugi del menija, kjer izberemo začetne in končne zamike pri zvonjenju (na sliki 4.3). Uporabljajo se enake tipke kot pri prejšnjem delu. Ob končanju pritisnemo tipko *izberi* in vzorec zvonjenja se shrani v EEPROM.



Slika 4.3: Meni za vnos zamikov

Meni je realiziran s pomočjo komponent:

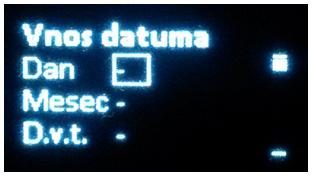
- oznaka za prikaz naslova menija in številk zvonov,

- krožna izbira za izbiro mesta v melodiji,
- celo število za izbiro odmika od začetka in konca zvonjenja ter
- horizontalni splošen drsni seznam za grupiranje komponent, ki niso mogle biti hkrati prikazane na zaslonu zaradi omejene velikosti zaslona.

4.3.3 Vnos časa zvonjenja

Meni za vnos časa zvonjenja je razdeljen na 5 delov, saj bi bil sicer nepregleden.

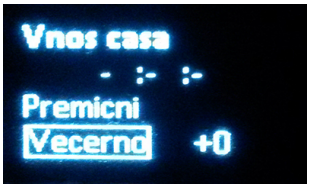
V prvem delu menija vnesemo datum zvonjenja. S tipkama *naprej* in *nazaj* se pomikamo med polji za vnos dneva, meseca, dneva v tednu, leta, premičnega datuma in odmika od vnešenega datuma. Vrednost polj spreminjamo s tipkama *gor* in *dol*, meni pa avtomatsko prepreči neveljavne vnose. V kolikor izberemo odmik od vnešenega datuma, se pojavijo še polja za vnos podrobnosti o izbranem odmiku. Ko končamo z vnašanjem, pritisnemo tipko *izberi*, da se pomaknemo na naslednji del.



Slika 4.4: Meni za vnos datuma zvonjenja

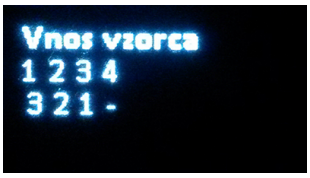
V drugem delu menija vnesemo čas zvonjenja. S tipkama *naprej* in *nazaj* se pomikamo med polji za vnos ure, minute, sekunde in premičnega časa (sončni zahod in sončni vzhod). S tipkama *gor* in *dol* izberemo vrednosti polj, neveljavne kombinacije pa so avtomatsko preprečene. V koliko izberemo premičen čas, se pojavi polje za vnos odmika od tega časa. Ko končamo z vnašanjem, pritisnemo tipko *izberi*, da se pomaknemo na naslednji del.

V tretjem delu menija izberemo vzorec zvonjenja. S tipkama *gor* in *dol* se pomikamo med vzorci zvonjenja, ki so razdeljeni na prikaz melodije in prikaz zamikov. S tipkama *naprej* in *nazaj* se pomaknemo na dele, ki niso



Slika 4.5: Meni za vnos ure zvonjenja

mogli biti prikazani zaradi omejene velikosti zaslona. Ko izberemo vzorec zvonjenja, pritisnemo tipko *izberi*, da se pomaknemo na naslednji del.



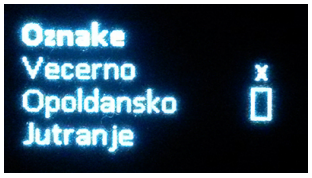
Slika 4.6: Meni za izbiro vzorca zvonjenja

V četrtem delu menija vnesemo trajanje zvonjenja. S tipkama *gor* in *dol* povečujemo ali zmanjšujemo sekunde, ki se sproti avtomatsko pretvarjajo v minute. Trajanje zvonjenja je navzdol omejeno z izbiro vzorca zvonjenja, saj morajo biti upoštevani vnešeni zamiki. Ko vnesemo trajanje zvonjenja, pritisnemo tipko *izberi*, da se pomaknemo na naslednji del.

Slika 4.7: Meni za vnos trajanja zvonjenja

V zadnjem delu menija lahko vnosu dodamo oznake (zastavice). Več o zastavicah in njihovem pomenu je opisano v poglavju 5. Med zastavicami se pomikamo s tipkama *naprej* in *nazaj*, zastavico pa označimo kot aktivno ali neaktivno s tipkama *gor* in *dol*. Ko smo končali z vnosom zastavic, pritisnemo tipko *izberi*, da vnešene podatke shranimo.

Za realizacijo menija so uporabljene komponente:



Slika 4.8: Meni za vnos oznak zvonjenja

- oznaka za prikaz naslova menija, opise polj in ločilne pike ter dvopičja,
- krožna izbira za izbiro dneva, meseca, leta, premičnega datuma, odmika od datuma, dneva v tednu, ure, minute, sekunde, premičnega časa in dodajanje oznak,
- celo število za vnos trajanja, odmika od premičnega časa in odmika od datuma v dnevih,
- splošen drsni seznam za grupiranje komponent menijev, ki niso mogle biti v celoti prikazane na zaslonu zaradi omejene velikosti zaslona in
- drsni trak za lažjo orientacijo v menijih, ki niso v celoti prikazani na zaslonu.

4.3.4 Nastavitev ure

V meniju za nastavitev ure (na sliki 4.9) lahko nastavljamo uro sistema. S tipkama *naprej* in *nazaj* se pomikamo med polji za vnos ure, minute in sekunde, s tipkama *gor* in *dol* pa nastavimo željeno vrednost polja.

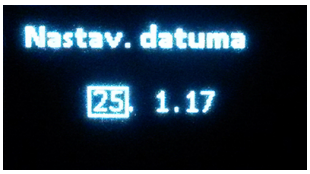
Slika 4.9: Meni za nastavitev ure

Meni je realiziran s pomočjo komponent:

- oznaka za prikaz naslova menija in dvopičij ter
- celo število za izbiro ure, minute in sekunde.

4.3.5 Nastavitev datuma

V meniju za nastavitev datuma (na sliki 4.10) lahko nastavljamo datum sistema. S tipkama *naprej* in *nazaj* se pomikamo med polji za vnos dneva, meseca in leta, s tipkama *gor* in *dol* pa nastavimo željeno vrednost polja.



Slika 4.10: Meni za nastavitev datuma

Meni je realiziran s pomočjo komponent:

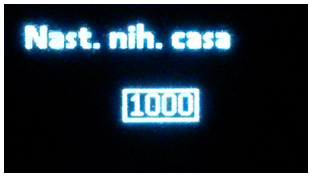
- oznaka za prikaz naslova menija in pik ter
- celo število za izbiro dneva, meseca in leta.

4.3.6 Nastavljanje nihajnega časa

V meniju za nastavljanje nihajnega časa (na sliki 4.11) lahko nastavljamo čas nihaja zvonov v milisekundah. S tipkama *gor* in *dol* nastavimo izmerjen čas nihaja, ob končanju pa pritisnemo tipko *izberi*, da potrdimo vnos.

Meni je realiziran s pomočjo komponent:

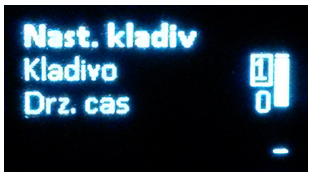
- oznaka za prikaz naslova menija in
- celo število za vnos nihajnega časa.



Slika 4.11: Meni za nastavljanje nihajnega časa

4.3.7 Nastavljanje parametrov kladiv

V meniju za nastavljanje parametrov kladiv lahko nastavljamo posamezne parametre, povezane z delovanjem kladiv. S tipkama *naprej* in *nazaj* se pomikamo med poljema za izbiro kladiva in vnos časa trajanja držanja signala v aktivnem stanju, s tipkama *gor* in *dol* pa spreminjamo vrednost v izbranem polju. Ob končanju pritisnemo tipko *izberi* za potrditev vnosa.



Slika 4.12: Meni za nastavljanje parametrov kladiv

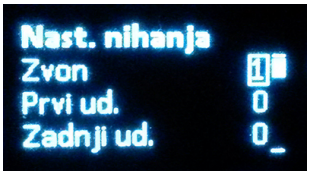
Meni je realiziran s pomočjo komponent:

- oznaka za prikaz naslova menija in naslovov polj,
- krožna izbira za izbiro kladiva,
- celo število za vnos časa trajanja držanja signala v aktivnem stanju,
- splošen drsni seznam za grupiranje komponent in
- drsni trak za lažjo orientacijo v meniju.

4.3.8 Nastavljanje parametrov nihanja zvonov

V meniju za nastavljanje parametrov nihanja zvonov lahko nastavljamo posamezne parametre, povezane z nihanjem zvonov. S tipkama *naprej* in *nazaj*

se pomikamo med polji za izbiro zvona, vnos časa trajanja od zagona do prvega udarca kemblja, vnos časa trajanja od ustavitve do zadnjega udarca kemblja, vnos popravka takta in vnos hitrosti motorja, s tipkama *gor* in *dol* pa spreminjamo vrednost v izbranem polju. Ob končanju pritisnemo tipko *izberi* za potrditev vnosa.



Slika 4.13: Meni za nastavljanje parametrov nihanja

Meni je realiziran s pomočjo komponent:

- oznaka za prikaz naslova menija in naslovov polj,
- krožna izbira za izbiro zvona,
- celo število za vnos ostalih parametrov,
- splošen drsni seznam za grupiranje komponent, ki niso mogle biti v celoti prikazane na zaslonu zaradi omejene velikosti zaslona in
- drsni trak za lažjo orientacijo v meniju.

4.3.9 Domači zaslon

Domači zaslon (na sliki 4.14) prikazuje trenutno uro in datum, ko je avtomat v stanju mirovanja. Za vstop v glavni meni pritisnemo tipko *izhod*, enako storimo, če se želimo vrniti na domači naslon.

Prikaz je realiziran s pomočjo komponente poljubno.

Slika 4.14: Domači zaslon

Poglavje 5

Shranjevanje urnika zvonjenja

Možnost shranjevanja urnika zvonjenja je nujna, če želimo uporabniku omogočiti kasnejše prenastavljanje avtomata po svojih željah in potrebah, ki se bodo verjetno skozi čas spreminjale. Pomembno je torej načrtovati avtomat z mislijo na to, da bo na nek način potrebno shraniti nastavitve avtomata in urnik zvonjenj, kar terja nekatere prilagoditve pri načrtovanju.

Navade zvonjenja se razlikujejo po regijah in krajih, zato je nemogoče vnaprej predvideti vse potrebe uporabnika avtomata. Uporabniku smo poskusili ponuditi čim več možnosti pri vnosu urnika, če pa je kasneje ugotovljeno, da nekatere funkcionalnosti, ki jih bi uporabnik potreboval, manjkajo, jih je mogoče naknadno implementirati.

5.1 Opis običajnih potreb

Da si bo moč lažje predstavljati posamezne odločitve pri načrtovanju avtomata, smo na tem mestu predstavili običajen urnik zvonjenja naključne cerkve v osrednjeslovenski ali gorenjski regiji.

Posamezno zvonjenje lahko razdelimo na dva logična sklopa:

- čas zvonjenja (ura in datum) in
- način zvonjenja (kako se bo zvonilo, npr. kateri zvonovi so uporabljeni,

v kakšnem vrstnem redu zvonijo, v kakšnem vrstnem redu se poženejo in s kakšnimi časovnimi razmiki itn.) [4, str. 27].

Poimensko lahko najpogostejša zvonjenja razdelimo na sledeč način:

Jutranje zvonjenje

Zvonjenje, ki se izvaja vsak dan ob 7. uri. Če ima cerkev tri zvonove, to delo izvaja veliki zvon, če pa ima štiri zvonove, to delo med delavnikom in ob sobotah izvaja srednje-veliki zvon, ob nedeljah pa veliki zvon. V primeru petih ali več zvonov je izbira zvona, ki se uporabi, različna.

Opoldansko zvonjenje

To zvonjenje je zelo podobno jutranjemu zvonjenju, le da se izvaja ob 12. uri, v krajih ob avstrijski meji na Koroškem in Štarjerskem pa ob 11. uri.

Večerno zvonjenje

Večerno zvonjenje se izvaja vsak dan približno petnajst minut do pol ure po sončnem zahodu. V primeru treh zvonov začne zvoniti srednji zvon. Ko le-ta konča, sledita dva udarca premolka in nato zazvoni mali zvon. Na enak način sledi še veliki zvon. Ponekod sta vlogi velikega in srednjega zvona obrnjeni ali pa je en zvon izpuščen, v nekaterih primerih tudi ponovljen. Če ima cerkev štiri zvonove, se med tednom zvoni z manjšimi tremi, ob nedeljah (in ponekod tudi ob sobotah) pa se večja dva zvonova zamenjata. V primeru petih ali več zvonov je izbira zvonov, ki se uporabijo, različna. Pred prazniki se namesto zvonjenja pritrkava.

Mašno zvonjenje

Mašno zvonjenje se zvoni petnajst minut pred mašo. V primeru treh zvonov se za delavniške maše uporabita manjša dva zvonova hkrati, ob nedeljah

pa se jima pridruži veliki zvon. Zagon zvonov je postopen z opaznimi zamiki med zagoni. Če ima cerkev štiri zvonove, se za delavniške maše uporabi bodisi enak pristop kot pri treh zvonovih ali pa se doda še srednje-veliki zvon. Ob nedeljah zvonijo vsi zvonovi. V primeru petih ali več zvonov se veliki zvon lahko prihrani za večje praznike. Vrstni red udarjanja zvonov med zvonjenjem se zelo razlikuje med kraji. Navada veleva tudi, da ob nedeljah, eno uro pred mašo, zazvoni veliki zvon (v primeru treh ali štirih zvonov), da se verniki začnejo pripravljati za prihod k maši. Med prazniki zvonjenje zamenja pritrkavanje pred mašo in po maši, začetek maše pa naznani veliki zvon z nekaj udarci.

Ure in datumi svetih maš so zelo različni. V pomembnejših cerkvah so maše vsak dan po večkrat, medtem ko so ponekod le ob nedeljah ali celo enkrat letno. Pojavljajo se tudi datumi kot so npr. maša vsako zadnjo nedeljo v mesecu ali maša vsako nedeljo meseca maja. Pogost pojav je tudi prestavitev ure maše skupaj s premikom ure ob spremembi letnega časa.

Zvonjenje v spomin na Kristusovo smrt in trpljenje

V spomin na Kristusovo smrt in trpljenje se zvoni vsak petek ob 15. uri z velikim zvonom.

Delopust

To zvonjenje predstavlja poziv h končanju dela in pripravi na praznični dan. Zvoni se ob sobotah in dan pred praznikom ob 15. uri pozimi in 16. uri poleti. Običajno zvonijo vsi zvonovi hkrati, med zagoni zvonov je opazen časovni zamik. Vrstni red udarjanja zvonov se med kraji razlikuje. Ponekod na začetku kratek čas zvoni samo veliki zvon, nato se mu pridružijo ostali. Pred večjimi prazniki se namesto zvonjenja pritrkava (če se to ne stori namesto večernega zvonjenja).

Zvonjenje ob smrti

Cerkve imajo pogosto zvon imenovan navec, ki je namenjen le zvonjenju ob smrti prebivalca in se ne uporablja pri ostalih opravilih. V primeru smrti otroka se zvoni enkrat, v primeru smrti ženske dvakrat, v primeru smrti moškega pa trikrat zapored z razmiki približno ene minute.

Mrliško zvonjenje

Mrliško zvonjenje se zvoni, ko v domačem kraju pokojnik leži do svojega pogreba, bodisi v mrliški vežici ali doma, in se lahko ljudje od njega pridejo poslovit. Zvoni se pred ali po jutranjem, opoldanskem in večernem zvonjenju, običajno z vsemi zvonovi hkrati. Zvonovi se zaženejo hkrati, vrstni red udarjanja v melodiji pa se glede na kraj razlikuje. Podobno kot pri zvonjenju ob smrti se tudi tu zvoni glede na spol pokojnika enkrat, dvakrat ali trikrat.

Pogrebno zvonjenje

Pogrebno zvonjenje se uporablja, medtem ko gre pogrebni sprevod od cerkve do pokopališča. Zvoni se z vsemi zvonovi, ki začnejo zvoniti hkrati. Vrstni red udarjanja zvonov se tudi tu razlikuje med kraji.

Jutranjica

Ponekod obstaja navada, da se na praznični dan ob sončnem vzhodu pritrkava in tako naznani pomembnost prihajajočega dneva.

Še enkrat moramo omeniti, da so našteta zvonjenja tista, ki se najpogosteje pojavljajo. Glede na lokalne navade se lahko seznam precej spremeni.

Edini čas, ko se zvonjenje redno ne sme izvajati, je pred praznikom velike noči od približno četrtka zvečer do sobote zvečer (odvisno od časa svetih maš).

Cerkveni prazniki so dnevi, ko se spominjamo pomembnejših dogodkov v zgodovini katoliške cerkve. Nekateri prazniki imajo vsako leto enak datum, medtem ko so drugi gibljivi. Praktično vsi gibljivi prazniki so odvisni od pojava prve spomladanske polne lune. Za primer naštejmo nekaj takih praznikov in opišimo način izračuna njihovih datumov:

- velika noč - obeleževanje prvo nedeljo po spomladanski polni luni,
- pepelnica - obeleževanje 46 dni pred veliko nočjo,
- cvetna nedelja - 6. postna nedelja,
- veliki četrtek - četrtek pred veliko nočjo,
- bela nedelja - nedelja po veliki noči,
- vnebohod - 40. dan po veliki noči,
- binkošti - 50. dan po veliki noči,
- sv. Trojica - nedelja po binkoštih,
- telovo - drugi četrtek po binkoštih,
- presveto srce Jezusovo - 8 dni po telovem,
- brezmadežno srce Marijino - 3. sobota po binkoštih in
- nedelja Kristusa, kralja vesolja - nedelja pred prvo adventno nedeljo.

Poseben praznik v vsakem kraju je god zavetnika cerkve. Ta dan se lahko obeležuje bodisi na prvo nedeljo po dejanskem godu svetnika, pred godom svetnika ali pa na najbližjo nedeljo godu svetnika. V redkih primerih se lahko nedelja, na katero se obeležuje god zavetnika cerkve, prekriva s kakšnim drugim praznikom. Tak primer je god sv. Simona in jude Tadeja, ki je 28. 10. Če je običajno obeležitev prvo nedeljo po godu, se le-ta lahko prekrije z dnevom spomina na mrtve. V takem primeru se obeleževanje običajno prestavi na drugo najbližjo nedeljo datumu goda.

Za praznike, tako kot za nedeljo, velja poseben red zvonjenja. V cerkvah s štirimi zvonovi se za praznike pogosto uporabi kar nedeljski red zvonjenja, le da se pred svetimi mašami pritrkava. Če ima cerkev več kot štiri zvonove, se lahko uporaba največjega zvona prihrani za večje praznike.

V cerkvenem letu poznamo tudi dvoje obdobij, ki predstavljata čas priprave na dva najpomembnejša cerkvena praznika:

- 4 nedelje pred božičem se razteza obdobje imenovano advent in
- 40 dni pred veliko nočjo predstavlja obdobje posta.

V teh obdobjih se ponekod zvonjenje nekoliko spremeni, za primer navedimo ljubljansko stolnico, kjer v postnem obdobju v spomin na Kristusovo smrt in trpljenje zvoni veliki zvon namesto srednje-velikega zvona ter Mengeš, kjer v teh obdobjih namesto dveh zvonov k tedenski maši izjemoma kličejo trije zvonovi.

5.2 Shranjevanje vnosov

Za shranjevanje urnika zvonjenj smo implementacijo razdelili na dva dela, prvi del se ukvarja s časom, drugi pa z vzorcem zvonjenja. Za tak pristop smo se odločili, ker se večkrat zgodi, da več časovnih vnosov uporablja isti vzorec zvonjenja. To povzroči dve nevšečnosti, ki sta redundantni podatki v EEPROMu in potreba po vnašanju vzorca zvonjenja pri vsakem časovnem vnosu posebej.

5.2.1 Časovni vnosi

Najdba učinkovitega načina shranjevanja časa je zahtevala nekoliko bolj kompleksen način obravnave podatkov. Poskušali smo zaobjeti čim več različnih tipov časovnih vnosov. Za časovne vnose skrbi razred *TimeSave* skupaj s pomožno strukturo *TimeEntry*. Definirali smo naslednja polja:

Ura, minuta in sekunda

Navedena polja služijo izbiri časa v dnevu, ko naj se zvonjenje izvede. V prid varčevanju s prostorom sta lahko polji za uro in minuto uporabljeni tudi za izbiro posebnih časov, kot sta čas sončnega vzhoda in zahoda, ki morata biti izračunana. Ko je polje za uro večje od 23, se številka uporabi kot indeks posebnega časa. V takem primeru polje za minute dobi pomen odmika od tega časa.

Dan v tednu

Tedensko zvonjenje je mogoče vnesti z uporabo spremenljivke za dan v tednu. Njen nabor vrednosti je nekoliko razširjen, da olajša vnos nekaterih pogosteje uporabljanih kombinacij, kot so zvonjenje ob delavnikih, zvonjenje ob sobotah in nedeljah ter zvonjenje ob delavnikih in sobotah. V večini primerov deluje restriktivno z ožanjem izbora na izbrane dni v tednu, v nekaterih primerih, ki bodo opisani kasneje, pa lahko služi tudi drugim namenom.

Dan in mesec

Polji za dan in mesec omogočata letno ponovitev zvonjenja. Če je naveden tudi dan v tednu, je izbor še ožji. Uporabimo lahko tudi druge kombinacije, npr. izpustimo dan in zvonjenje se bo izvajalo le na določen mesec v letu, ko bo izbran dan v tednu itn.

Premakljivi prazniki so podprti preko polja za mesec, ko je le-ta večji od 12. Takrat število pomeni indeks posebnega datuma, ki se (običajno) mora izračunati, npr. datum velike noči.

Leto

Polje za leto lahko služi kot dodatna omejitev zvonjenja na enkratno izvedbo. Bolj pomemben pa je sekundarni pomen polja, ko izbrano število

preseže vrednost 99. Takrat polje dobi pomen indeksa posebnih funkcij, ki omogočajo vnos kompleksnejših datumov zvonjenj. Leto in posebne funkcije se ne uporabljajo hkrati, saj če vnašamo enkraten dogodek, gotovo vemo tudi točen datum le-tega. V avtomatu so implementirane sledeče funkcije:

- zvonjenje na n-ti izbrani dan v tednu po ali pred izbranim datumom (npr. druga nedelja po božiču),
- zvonjenje na n-ti dan po ali pred izbranim datumom (uporabno pri gibljivih datumih, npr. 50. dan po veliki noči),
- zvonjenje na n-ti izbrani dan v tednu v izbranem mesecu ali vsakem mesecu, šteto bodisi od začetka meseca naprej ali od konca meseca nazaj (npr. vsako drugo nedeljo v mesecu) in
- zvonjenje na najbližji izbrani dan v tednu izbranemu datumu (npr. nedelja, ki je bližje 25. 12.).

Polje za dan v tednu v funkcijah, ki ga uporabljajo, dobi pomen izbire namesto restrikcije. Za izbiro števila dni ali zaporedne številke dneva v tednu je vpeljano dodatno polje.

Zastavice

Polje z zastavicami omogoči dodajanje lastnosti posameznim vnosom. Definirane so naslednje zastavice:

- **Označevanje imenovanih zvonjenj** Ta skupina zastavic označi zvonjenje kot večerno, jutranje, opoldansko, mašno itn. To se lahko uporabi pri izvedbi mrliškega zvonjenja, ki se izvaja v povezavi z določenimi zvonjenji, pa tudi pri možnosti izključitve mašnega zvonjenja v primeru, ko maše odpadejo.
- **Izvedba kljub prazniku** Z uporabo paketov za praznike se običajno zvonjenje, ki bi se izvajalo na ta dan, ne izvede. Včasih je zaželjeno,

da se nekatere izjeme, kot je npr. zvonjenje v spomin na Kristusovo smrt in trpljenje, vseeno izvedejo, čemur je namenjena obravnavana zastavica.

- **Povozi bitje ure** Če bi vnos moral čakati, da najprej odbije ura, se bitje ure ne izvede.
- **Premikaj z letnim časom** Pri premiku ure zaradi spremembe letnega časa, se premakne tudi vnos.
- **Izvedi enkrat** Vnos se po izvedbi izbriše.

Trajanje

V tem polju je navedeno trajanje izvedbe zvonjenja v sekundah.

Vzorec

Polje vzorec hrani indeks vzorca, ki vsebuje podrobnosti o načinu izvedbe zvonjenja. Ker časovni vnos ne more obstajati brez vzorca, se to polje hkrati uporablja za preverjanje veljavnosti časovnega vnosa.

Na voljo so funkcije za nastavljanje in pridobivanje vseh naštetih parametrov. Vsa polja je mogoče shraniti v 13 B dolg seznam, ki se ga shrani v EEPROM, enako velja za obraten proces. Za ta namen in za brisanje so na voljo funkcije *naloži*, *shrani* in *izbriši*, ki delujejo z uporabo indeksov, kjer indeks i predstavlja zaporedno številko vnosa v EEPROMu, $0 \leq i < n$. n predstavlja največje dovoljeno število vnosov, ki so lahko hkrati shranjeni v EEPROMu. Za shranjevanje je na voljo funkcija, ki poišče prvi prazen prostor v EEPROMu, kamor se lahko shrani nov vnos. Na voljo je tudi funkcija, ki izbriše vse vnose hkrati.

Ko je potrebno preveriti, ali je čas, da se vnešeno zvonjenje izvede, se vrednosti polj po potrebi pretvori oz. izračuna na sledeči način:

- neuporabljena polja za dan, mesec in leto se zapolnijo z vrednostmi trenutnega časa,
- izračuna se prioriteta vnosa glede na pogostost ponavljanja (manj pogosti vnosi imajo prednost),
- v primeru uporabe posebnih datumov se le-ti izračunajo glede na trenutno leto,
- v primeru uporabe posebnih časov se le-ti izračunajo glede na trenutni datum in
- v primeru uporabe posebnih funkcij se le-te izvršijo nad podatki, vpisanimi v strukturo *TimeEntry*.

Rezultat teh operacij je vedno v strukturo *TimeEntry* vpisan absoluten datum, ki se ga nato primerja trenutnim datumom.

5.2.2 Vnosi vzorcev zvonjenja

Shranjevanje načina zvonjenja je implementirano v razredu *RingSample*. Shranjeni so sledeči parametri:

- melodija zvonjenja,
- zamik zagona na začetku zvonjenja in
- zamik ustavitve na koncu le-tega.

Implementirane so funkcije za nastavljanje in pridobivanje vseh naštetih parametrov.

Za shranjevanje vrstnega reda zvonov v melodiji je uporabljen seznam, ki za vsak zvon vsebuje njegovo pozicijo v melodiji. Polja predstavljajo zvonove, vpis v polje pa mesto v melodiji. Uporaba zapisa je razložena v poglavju 7.

Zamik zagona omogoča, da zvon začne zvoniti s poljubnim zamikom od začetka zvonjenja. To se začne ob prvem udarcu zvona, ki ima ničelni odmik. Ta funkcija je zelo pogosto uporabljana pri razlikovanju med mrliškim

zvonjenjem in mašnim zvonjenjem, saj pri mrliškem vsi zvonovi začnejo zvoniti hkrati, pri mašnem pa se zaganjajo postopoma. Zamiki so shranjeni v seznamu, kjer polje predstavlja zvon, vnos v polje pa zamik v sekundah od začetka zvonjenja.

Zamik pri ustavljanju je realiziran zelo podobno kot zamik pri zagonu. Omogoča, da se zvon ustavi poljubno število sekund pred koncem zvonjenja, le-to pa se konča, ko zadnjič udari zvon z ničelnim odmikom. Tako je moč doseči postopno ustavljanje zvonov, kar lahko naredi zaključek zvonjenja lepši in bolj dramatičen.

Vse podatke je mogoče za namen shranjevanja pretvoriti v seznam bajtov (in obratno), njegova dolžina pa se razlikuje glede na uporabljeno število zvonov, ki se določi pred prevedbo programa. Za nalaganje, shranjevanje in brisanje so na voljo funkcije *naloži*, *shrani* in *izbriši*, ki kot parameter prejmejo indeks vzorca i, shranjenega v EEPROMu, $0 \leq i < n$, kjer je n največje dovoljeno število vzorcev, ki so lahko hkrati shranjeni v EEPROMu. Za lažje shranjevanje na novo vnešenih vzorcev je na voljo funkcija, ki poišče prvi prazen prostor v EEPROMu. Na voljo je tudi funkcija, ki izbriše vse vzorce hkrati.

Ob naložitvi vzorca iz EEPROMa se za lažjo obdelavo podatkov sestavi tudi vrstni red zagona zvonov. Le-tega se dobi z uporabo algoritma za razvrščanje nad podatki o zamikih zagona zvonov.

Poglavje 6

Upravljanje z zvonovi

V sledečem poglavju so opisane entitete, ki upravljajo z različnimi načini uporabe zvona, kot sta zvonjenje in udarjanje s kladivom.

6.1 Zvonjenje

6.1.1 Upravljalec zvona

Upravljalec zvona poenostavi uporabo zvona v namene zvonjenja. Implementiran je v razredu *Bell*. V sebi hrani tri ključne sklope funkcij:

- funkcije za nalaganje in shranjevanje podatkov iz EEPROMa,
- funkcija za dostop do podentitete *krmilnik nihanja* in
- funkcije za dostop do podatkov o posameznem zvonu.

Funkcije za nalaganje in shranjevanje podatkov iz EEPROMa skrbijo za prenos vseh parametrov entitete in podentitet iz in v EEPROM, vključno s serializacijo pred prenosom v EEPROM in deserializacijo po prenosu iz EEPROMa. Podatki so v EEPROMu shranjeni z uporabo indeksov, ki lahko zavzamejo vrednosti $0 \leq i < n$, kjer je n število zvonov, ki so zmožni nihanja. Osnovni funkciji *naloži* in *shrani* v entiteto naložita podatke o zvonu i, na

voljo pa so tudi funkcije, ki naložijo podatke o vseh zvonovih in izbrišejo podatke o vseh zvonovih.

Funkcija za dostop do podentitete *upravljalec nihanja* omogoča upravljanje z nihanjem zvona. *Upravljalec nihanja* vsebuje funkciji za zagon in ustavitev, kar omogoča zagon ali zaustavitev zvona z enim preprostim ukazom. Podrobnejši opis delovanja *upravljalca nihanja* sledi v nadaljevanju.

Funkcije za dostop do podatkov omogočajo nastavljanje in pridobivanje naslednjih parametrov o posameznem zvonu:

- nihajni čas, ki predstavlja čas enega nihaja zvona, iz ene skrajne pozicije v drugo,
- čas od zagona do prvega udarca, ki predstavlja pretečen čas od ukaza za zagon zvona do prvega udarca kemblja ob rob zvona,
- čas od zaustavitve do zadnjega udarca, ki predstavlja čas od ukaza za zaustavitev zvona do zadnjega udarca kemblja,
- popravek takta, ki predstavlja izmerjen razmik med časom, ko bi kembelj moral udariti in časom, ko kembelj dejansko udari ob rob zvona in
- hitrost motorja, ki predstavlja hitrost vrtenja motorja, ki poganja zvon.

Uporaba nekaterih omenjenih parametrov je opisana v poglavju 7.

6.1.2 Upravljalec izhodnega signala

Upravljalec izhodnega signala skrbi za nastavitev izhodov in časovnikov mikrokrmilnika, da proizvajajo potreben izhodni signal za krmiljenje motorja, ki poganja zvon. Implementiran je v razredu *PWM_Driver*.

Za krmiljenje motorja je uporabljena pulzno-širinska modulacija (*PWM*) frekvence 21 kHz, kar je za človeško uho neslišno. Časovniki so nastavljeni na centrirano modulacijo (*center-aligned pwm*), kar pomeni, da se časovnik, ko doseže najvišjo dovoljeno vrednost, ne ponastavi na 0, temveč se začne

odštevati in tako kroži med 0 in najvišjo dovoljeno vrednostjo. Tak način je boljši za nadzor motorjev, saj povzroča manj motenj.

Za generiranje pulzno-širinske modulacije je uporabljen način z dvema kanaloma: osnovnim kanalom in njegovim inverzom (*locked antiphase*) [3]. Če osnovni kanal nastavimo na najvišjo vrednost, ki jo časovnik lahko doseže, se motor z največjo možno hitrostjo vrti v pozitivno smer. Takrat je signal na tem kanalu aktiven 100 % časa, medtem ko je na njemu inverznem kanalu aktiven 0 % časa. Če osnovni kanal nastavimo na 0, se motor z največjo možno hitrostjo vrti v negativno smer. Takrat je signal na tem kanalu aktiven 0 % časa, medtem ko je na njemu inverznem kanalu aktiven 100 % časa. Kadar pa je osnovni kanal aktiven 50 % časa, je motor ustavljen.

6.1.3 Upravljalec nihanja

Upravljalec nihanja skrbi za vse, kar je povezano z nihanjem zvona. Od ukaza za začetek do ukaza za konec skrbi, da motor niha zvon z vnešenim nihajnim časom in hitrostjo. Nihajni čas se v avtomat vnese po izmeri naravnega nihajnega časa zvona pri predvidenem kotu nihanja, pravilno nastavljena hitrost motorja pa je sinhronizirana s hitrostjo zvona, saj v nasprotnem primeru zvon bodisi ne zaniha na želen kot ali pa zaniha previsoko, kar ga lahko uniči bodisi ob trku z oviro ali ob premočnem udarcu kemblja. *Upravljalec nihanja* je implementiran v razredu *Swinger*. Za generiranje izhodnega signala uporablja *upravljalca izhodnega signala*.

Nihaj zvona je razdeljen v tri faze:

1. Prva faza se začne, ko je zvon v eni izmed skrajnih leg. Motor in zvon imata v tej točki hitrost 0. Zvon iz te točke sili proti tlom zaradi sile gravitacije in njegova hitrost narašča, ko se bliža nevtralni legi. Motor to gibanje poskuša oponašati s postopnim povečevanjem hitrosti.

2. Druga faza se začne tik pred nevtralno lego in konča tik po nevtralni legi. V tej fazi je zvon najhitrejši, zato se motor tu vrti z vnešeno hitrostjo, ki je tudi najvišja hitrost, ki jo motor lahko doseže.

3. Tretja faza se začne tik po nevtralni legi, ko zvon zaradi svoje kinetične energije zaniha v drugo smer in se začne postopoma ustavljati. Motor tu postopoma upočasnjuje do popolne ustavitve, ko zvon zopet zamenja smer in ponovno se prične faza 1.

V primeru, ko je motor prešibak, da bi zagnal ustavljen zvon, dovolj močan pa za ohranjanje zvona v polnem nihanju, je mogoče nihajni čas postopoma povečevati, ko se nihajni kot zvona povečuje (saj je nihajni čas vezan na kot nihanja zvona). To razbremeni motor ob zagonu, vendar v tem avtomatu ni implementirano, povzroči pa lahko tudi daljše zahajanje zvona v takt z ostalimi zvonovi.

Za izvedbo naštetih faz se vnaprej izvedejo sledeči izračuni:

1. Nihajni čas se razdeli na odseke.

$$t_{\text{odsek}} = \frac{t_{\text{nihaj}}}{n} \;\Big|\; n - \text{st. odsekov} \tag{6.1}$$

   Odsek predstavlja najmanjšo enoto, ko je motorju mogoče spreminjati pospeševanje ali upočasnjevanje. Najbolj idealno bi bilo popolnoma slediti naravni hitrosti zvona, kar bi zahtevalo vnaprejšnjo izmero hitrosti v vsaki točki s čim višjo frekvenco vzorčenja in najti funkcijo, ki se tako dobljenim podatkom čim tesneje prilega. Ker se za ta način potrebuje natančen senzor gibanja, smo se za potrebe diplomskega dela zadovoljili z grobim približkom: nihajni čas se razdeli na 5 odsekov.

2. Prej opisana prva faza zavzema prva dva odseka, ko se konstantno pospešuje, druga faza tretji odsek, ko je hitrost konstantna, tretja faza pa četrti in peti odsek, ko se konstantno upočasnjuje. Ker se bo pospešek spreminjal ob prehodih med fazami, se izračunajo meje med fazami.

$$\begin{aligned} T_{\text{meja 1}} &= \lfloor 2 * t_{\text{odsek}} \rfloor - 1 \\ T_{\text{meja 2}} &= \lfloor 4 * t_{\text{odsek}} \rfloor - 1 \end{aligned} \tag{6.2}$$

3. Ker mikrokrmilnik ne more zvezno povečevati hitrosti, temveč digitalno, si je potrebno izbrati časovno enoto, na katero se hitrost spreminja. Zadostuje časovna enota velikosti 1 milisekunde, ker je z njo lažje

operirati in ker z njo dobimo dovolj gladke prehode hitrosti za potrebe tega diplomskega dela. V ta namen se izračuna potreben korak za dosego želene hitrosti glede na vnešen nihajni čas, tako da se deli želeno ciljno hitrost (ki je v tem primeru najvišja hitrost motorja) z dvakratnikom časa trajanja odseka, kar ustreza dolžini prve in tretje faze. Omeniti je potrebno, da so za časovne enote uporabljene milisekunde, za enoto hitrosti pa širina pulza pri pulzno-širinski modulaciji.

$$\Delta = \frac{v_{max}}{2 * t_{odsek}} \tag{6.3}$$

v_{max} predstavlja najvišjo hitrost motorja v obliki odstopanja od točke v_0, kjer je signal pri pulzno-širinski modulaciji aktiven 50 % časa. Odstopanje v pozitivno ali negativno smer od točke v_0 diktira smer motorja.

Ves čas nihanja zvona nato upravljalec nihanja prehaja med omenjenimi tremi fazami:

1. Števec nihajnega časa, ki se povečuje vsako milisekundo, je na začetku enak 0, izhodni signal pa na nevtralni točki v_0, ko je hitrost 0. Hitrost motorja se ob vsakem povečanju števca povečuje od 0 do v_{max} s prištevanjem koraka Δ k nevtralni točki v_0. Če je korak Δ pozitiven, se bo motor vrtel v pozitivno smer, če pa je negativen, se bo motor vrtel v negativno smer.

2. Ko se doseže mejo druge faze, ostane hitrost konstantna na v_{max}, kar za izhodni signal pomeni $v_0 + v_{max}$ v primeru pozitivne smeri ali $v_0 - v_{max}$ v primeru negativne smeri.

3. Ob meji s tretjo fazo se začne hitrost zmanjševati z odštevanjem koraka in doseže 0 ob ponastavitvi števca na 0. Takrat je izhodni signal zopet enak v_0. Na tej točki se zamenja smer motorja z zamenjavo predznaka koraka Δ in ponovi se korak 1.

6.2 Upravljalec kladiva

Da se drugim entitetam, ki uporabljajo kladiva, ni potrebno ukvarjati z nadzorom posameznih kladiv, za to skrbi entiteta *upravljalec kladiva*, ki je implementiran v razredu *Hammer*. Ker so lahko nekateri zvonovi statični (ne zvonijo), je to samostojna entiteta.

Upravljalec kladiva vsebuje funkcijo, ki sproži udarec kladiva po zvonu, funkcije za naložitev in shranjevanje parametrov kladiva ter funkcije za nastavljanje in pridobivanje parametrov. Podobno kot pri entiteti upravljalec zvona ima impementirani funkciji *naloži* in *shrani*, ki delujeta preko indeksov $0 \leq i < n$, kjer je n število kladiv. Implementirana je tudi funkcija za naložitev podatkov vseh kladiv ali izbris podatkov vseh kladiv.

Potrebni parametri se razlikujejo glede na tip kladiva. Najosnovnejša in pogosto uporabljana kladiva so elektromagneti, ki potrebujejo le čas ohranjanja izhodnega signala v aktivnem stanju. Ta čas mora biti dovolj dolg, da tuljava uspe izsteliti železno jedro, ki udari ob zvon, vendar dovolj kratek, se jedro nato čim hitreje pomakne nazaj v tuljavo.

Poglavje 7

Izvedba zvonjenja

V sledečem poglavju so po vrsti opisane entitete, ki omogočajo izvedbo zvonjenja. Za prehajanje med različnimi stanji je uporabljen končni avtomat, ki ima definirana naslednja stanja:

- mirovanje,
- uporabnik v menijih,
- bitje ure v poteku,
- premor med zvonjenjem in bitjem ter
- zvonjenje v poteku.

7.1 Izvajalec opravil

Izvajalec opravil vsako sekundo preverja, ali je čas za izvajanje zvonjenja in bitja ure. To počne le, ko je avtomat v stanju *mirovanja*. Implementiran je v razredu *Scheduler*.

Serializirani časovni vnosi se v paketih po 10 začasno prenesejo iz EEPROMa v pomnilnik mikrokrmilnika, kjer se jih iz bitnih polj sestavi nazaj v objekte *TimeSave*. Veljavne vnose se nato pretvori v strukturo *TimeEntry*, kar omogoči primerjavo s trenutnim časom (postopek pretvorbe je opisan v

poglavju 5). V primeru ujemanja s trenutnim časom se vnos začasno shrani. Če se kateri izmed nadaljnjih vnosov prav tako ujema s trenutnim časom, se shrani tistega, ki ima višjo prioriteto.

Po končanju pregleda časovnih vnosov se (v primeru najdbe vnosa, ki se ujema s trenutnim časom) iz EEPROMa naloži vzorec zvonjenja, katerega indeks je zapisan v časovnem vnosu, in se ga skupaj s trajanjem zvonjenja poda *izvajalcu zvonjenja* v pripravo. Če je imel časovni vnos označeno zastavico za enkratno izvedbo, se ga na tej točki iz EEPROMa izbriše.

V kolikor pri časovnem vnosu ni bila označena zastavica *povozi bitje ure*, se na tem mestu preveri, ali je potrebno izvesti tudi bitje ure. To se stori s klicem *izvajalca bitja*. V primeru bitja ure se stanje avtomata prestavi na *bitje ure v poteku* in izvajalec bitja izvede bitje ure, sicer pa se stanje prestavi direktno na *zvonjenje v poteku*, v kolikor je na sporedu zvonjenje. Po koncu bitja ure *izvajalec bitja* spremeni stanje avtomata v *premor med zvonjenjem in bitjem*. V primeru, da je na sporedu zvonjenje, se naredi premor med bitjem in zvonjenjem, nato pa se stanje avtomata prestavi na *zvonjenje v poteku*, na zaslon pa se izpiše napis "zvonjenje". Če na sporedu ni zvonjenja, se stanje spremeni v *mirovanje*.

7.2 Izvajalec bitja

Izvajalec bitja skrbi za izvedbo bitja ure. Bitje ure se izvaja na vsak četrt ure, število izvedb n pa se določi po naslednjem ključu:

min.	n
15	1
30	2
45	3
0	4

(7.1)

Izbrana melodija, ki predstavlja četrt ure (v našem primeru sta to zaporedna udarca kladiv po manjših dveh zvonovih) se ob ujemanju minut izvede n-krat. V primeru polne ure (minute so enake 0) se izvede tudi bitje ure, kar

predstavljajo zaporedni udarci kladiva po velikem zvonu tolikokrat, kolikor je ura v 12-urnem sistemu. Za udarce kladiv izvajalec bitja uporablja *upravljalca kladiv*. Po končanem bitju izvajalec bitja spremeni stanje avtomata v *premor med zvonjenjem in bitjem*.

Izvajalec bitja je implementiran v razredu *Striker*.

7.3 Izvajalec zvonjenja

Ko *izvajalec opravil* poda zvonjenje v izvajanje, delo prevzame *izvajalec zvonjenja*, ki je implementiran v razredu *Ringer*.

V fazi priprave se glede na podatke o

- času, ki ga posamezen zvon potrebuje od zagona do prvega udarca,
- času, ki ga posamezen zvon potrebuje od začetka zaustavitve do zadnjega udarca,
- zamikih zagonov in ustavljanj, zapisanih v vzorcu zvonjenja in
- trajanju zvonjenja

izračunajo časi dejanskih zagonov in ustavljanj zvonov. Upoštevati je namreč treba dejstvo, da zvon ne začne zvoniti nemudoma, prav tako pa se nemudoma tudi ne ustavi, časi od zagona do prvega udarca in od ustavitve do zadnjega udarca pa se med zvonovi lahko tudi razlikujejo. Izračun dejanskih časov mogoča, da so kljub tem lastnostim zvonov upoštevani zamiki vnešeni v vzorec zvonjenja. Izračuni se izvedejo na sledeči način:

i: zaporedna številka zagona, zvon z nižjo številko se zažene prej
n: število uporabljenih zvonov

$$0 \leq i < n$$

1. Izračuna se dejanski čas trajanja zvonjenja, tako da se času trajanja zvonjenja prišteje čas zagona zvona, ki bo prvi pričel zvoniti (čas tišine).

$$t_{\text{zvonj. total}} = t_{\text{zvonj.}} + t_{\text{zaganj. 0}} \tag{7.2}$$

2. Za vsak zvon se seštejeta čas od zagona do prvega udarca zvona, ki bo prvi začel zvoniti, in zamik zagona trenutno obravnavanega zvona od začetka zvonjenja. Od rezultata se nato odšteje čas od zagona do prvega udarca obravnavanega zvona.

$$T_{\text{zagona } i} = t_{\text{zaganj. } 0} + t_{\text{zac. zamik } i} - t_{\text{zaganj. } i} \tag{7.3}$$

3. Če ima kateri izmed zvonov, ki se ne zaženejo prvi, izredno dolg čas od zagona do prvega udarca, se lahko zgodi, da ga je treba zagnati pred zvonom, ki mora udariti prvi. To se preveri tako, da se seštevek iz prejšnje točke (rezultat dobljen pred odštevanjem) odšteje od časa zagona do prvega udarca obravnavanega zvona. V primeru, da je rezultat pozitiven in večji od rezultata pri prejšnjih obravnavanih zvonovih, se ga zabeleži.

$$\begin{gathered} t_{\text{razsir. } i} = t_{\text{zaganj. } i} - (t_{\text{zaganj. } 0} + t_{\text{zac. zamik } i}) \\ t_{\text{razsir. max}} = \max\left[t_{\text{razsir. } i}\right] \end{gathered} \tag{7.4}$$

4. Če je imel kateri koli zvon pri prejšnji točki rezultat večji od 0, se največjega izmed teh prišteje času zagona vseh zvonov in dejanskem času trajanja zvonjenja.

$$\begin{gathered} T_{\text{zagona } i} = T_{\text{zagona } i} + t_{\text{razsir. max}} \\ t_{\text{zvonj. total}} = t_{\text{zvonj. total}} + t_{\text{razsir. max}} \\ t_{\text{razsir. max}} > 0 \end{gathered} \tag{7.5}$$

5. Izračuna se čas ustavitve zvona, tako da od dejanskega časa zvonjenja odštejemo zamik ustavitve obravnavanega zvona in čas ustavljanja letega.

$$T_{\text{ustav. } i} = t_{\text{zvonj. total}} - (t_{\text{kon. zamik } i} + t_{\text{ustav. } i}) \tag{7.6}$$

Seznam z melodijo zvonjenja, ki je shranjen v vzorcu zvonjenja, se poda *vzdrževalcu takta* v pripravo. Vsak zvon, ki je uporabljen pri trenutnem zvonjenju, se označi kot uporabljen, ostali zvonovi pa so med zvonjenjem prezrti.

Medtem ko je avtomat v stanju *zvonjenje v poteku*, *izvajalec zvonjenja* konstantno preverja glede na prej izračunane čase, ali je čas za zagon posameznega zvona, njegovo zaustavitev in ali je čas zvonjenja potekel.

Ob zaključku zvonjenja se stanje avtomata spremeni nazaj v *stanje mirovanja*, na zaslonu pa se izriše domači zaslon.

7.4 Vzdrževalec takta

Naloga *vzdrževalca takta* je, da vsak vklop zvona po potrebi zakasni za tako dolgo, da bo udarjal na pravem (izbranem) mestu v melodiji. Da se to doseže, se pred zvonjenjem naredi nekaj izračunov:

i: zaporedna številka zvona
n: število zvonov
m: seznam, ki za vsak zvon vsebuje njegovo mesto v melodiji

$$0 \leq i < n$$

1. Največjemu mestu katerega koli zvona v melodiji, ki je uporabljen pri trenutnem zvonjenju, se prišteje 1 (prvo mesto je ničto mesto) in se ga vzame kot število mest v trenutni melodiji. Nihajni čas se deli s tem številom in tako se dobi razmik med udarci dveh sosednjih zvonov v melodiji.
$$t_{\text{razmik}} = \frac{t_{\text{nihaj}}}{\max[m_i] + 1} \tag{7.7}$$
2. Mesto vsakega zvona v melodiji se množi z v prejšnji točki izračunanim razmikom in tako dobi čas, ko je potrebno zvon zagnati, da bo udarjal na pravem mestu v melodiji.
$$T_{\text{zagona } i} = m_i * t_{\text{razmik}} \tag{7.8}$$
3. Času zagona vsakega zvona se prišteje popravek zagona, ki je bil v avtomat ročno vnešen za vsak zvon posebej (več v poglavju 6) in

omogoča, da se kljub mehanskim neenakostim med zvonovi dobi popoln takt.

$$T_{\text{zagona } i} = T_{\text{zagona } i} + t_{\text{popravek } i} \tag{7.9}$$

4. Če je pri prištevanju popravka čas zagona nekega zvona postal negativno število, se časom zagona vseh zvonov prišteje število, ki negativni čas zagona popravi na 0.

$$\begin{gathered} T_{\text{zagona } i} = T_{\text{zagona } i} + |\min\left[T_{\text{zagona } i}\right]| \\ |\min\left[T_{\text{zagona } i}\right]| < 0 \end{gathered} \tag{7.10}$$

Medtem ko je vsaj eden izmed zvonov aktiven (vklopljen), se *vzdrževalec takta* izvaja. Ker je zaželjeno, da so zvonovi vedno zagnani v isti smeri (ponekod statika zvonika prepoveduje, da bi vsi zvonovi zanihali v isto smer, saj to lahko ogrozi stabilnost stavbe), se števec nihajnega časa ponastavi na 0 šele ob dvakratniku nihajnega časa zvonov. Ko *vzdrževalec takta* zazna, da je bil nek zvon vklopljen s strani uporabnika ali *izvajalca zvonjenja*, počaka z dejanskim vklopom zvona, dokler števec nihajnega časa, ki konstantno teče od 0 do dvakratnika nihajnega časa, doseže izračunan čas zagona zvona. Takrat se zvon tudi dejansko vklopi.

Vzdrževalec takta je implementiran v razredu *Mill*.

Poglavje 8

Sklepne ugotovitve

V diplomskem delu smo predstavili, kako so se skozi zgodovino spreminjale avtomatizacije zvonjenja, od popolnoma mehanskih sistemov, ki so uporabljali mehanske ure in pogone z vrtečo gredjo, do današnjih računalniško vodenih rešitev, ki uporabljajo frekvenčno regulacijo za dosego popolnega nadzora nad motorji. Razložili smo pojme povezane z zvonjenjem zvonov in kakšni so običaji zvonjenja v slovenskih pokrajinah. Tako smo spoznali, da se v Sloveniji uporablja zvonjenje v taktu, ki zahteva drugačen pristop pri načrtovanju avtomata kot izven slovenskega prostora. Seznanili smo se z običajnimi urniki in načini zvonjenj, ki se pojavljajo v Sloveniji. Iz tega smo lahko nato ugotovili, katere funkcionalnosti mora podpirati sodoben avtomat za zvonjenje.

V nadaljevanju smo predstavili potek izdelave lastnega avtomata za zvonjenje. Jedro sistema predstavlja ploščica STM32F4 Discovery. Za trajno shranjevanje vnesenih podatkov smo uporabili EEPROM 24FC512. Da bi dosegli čim večjo točnost, smo za hranjenje ure realnega časa uporabili modul Chronodot, uporabniku pa omogočili vpogled v avtomat preko OLED prikazovalnika. Vse naštete naprave smo med seboj povezali preko vmesnika I^2C.

Za nastavljanje avtomata smo izdelali lasten uporabniški vmesnik, ki deluje z vnašanjem komponent v vsebnik. Le-ta skrbi, da se vse komponente

izrisujejo ter preko tipk prejemajo uporabnikove ukaze. Uporabnikov pritisk na tipko zajame instanca razreda, ki predstavlja fizično tipko in ga pretvori v ukaz, ki se nato poda aktivni komponenti.

Shranjevanje urnika zvonjenja smo razdelili na dve komponenti, časovno in izvedbeno. Obe morata biti zmožni serializacije, da se ju lahko trajno shrani. Časovna komponenta poleg statičnih datumov omogoča tudi vnos premičnih, izvedbena komponenta pa se ukvarja z melodijo zvonjenja ter zamiki zagonov in ustavljanj.

Posamezni elementi zvonjenja so predstavljeni z entitetami, ki imajo med seboj razdeljene zadolžitve. Tako entiteta *upravljalec zvona*, ki shranjuje parametre o zvonu, uporablja entiteto *upravljalec nihanja* za nadzor nad nihanjem zvona, le-ta pa uporablja entiteto *upravljalec izhodnega signala*, ki skrbi za nastavitev parametrov pulzno-širinske modulacije. Uporabo posameznega kladiva nadzoruje entiteta *upravljalec kladiva*.

Za izvedbo zvonjenja se uporablja končni avtomat, ki skrbi, da je vrstni red izvedbe vedno spoštovan in da vsaka entiteta lahko opravi svojo zadolžitev. Tako *upravljalec opravil* preverja, ali je čas za zvonjenje, samo zvonjenje pa preda v izvedbo *upravljalcu zvonjenja*. Če je vmes potrebno odbiti uro, to opravi *upravljalec bitja*. Za zagon zvonov v taktu skrbi *vzdrževalec takta*. Pojasnili smo tudi izračune, ki so potrebni za dosego te izvedbe.

Končni rezultat diplomskega dela je popolnoma funkcionalen avtomat za zvonjenje, primarno namenjen pogonu manjših zvonov, ki pa ga je z nekaj prilagoditvami elektronike mogoče uporabiti za pogon cerkvenih zvonov. Omogoča vnos katerega koli datuma in ure v letu, pa tudi vnos premičnih datumov, česar mnogi avtomati ne podpirajo. Podprto je zvonjenje v poljubni melodiji s poljubnimi zamiki zagona in ustavitve zvonov, za popolno upoštevanje vnesenih parametrov zvonjenja pa omogoča tudi vnos lastnosti posameznega zvona, kar omogočajo le najmodernejši avtomati.

Naslednje stopnje v razvoju avtomata bi bile dodajanje pritrkovalskih melodij, kar je pogosto rabljena funkcionalnost, vnos poljubne melodije bitja ure ter olajšanje vnosa prazničnega zvonjenja z uporabo paketov, kjer se

poljubnemu dnevu dodeli paket zvonjenja, ki vključuje vse potrebne vnose za zvonjenje preko celega dneva.

Literatura

[1] C++ Inheritance. [Online]. Dosegljivo: https://www.tutorialspoint.com/cplusplus/cpp_inheritance.htm. [Dostopano 25. 1. 2017].

[2] Load and save your Settings to the EEPROM. [Online]. Dosegljivo: http://playground.arduino.cc/Code/EEPROMLoadAndSaveSettings. [Dostopano 25. 1. 2017].

[3] Lock Anti-Phase Drive. [Online]. Dosegljivo: http://www.modularcircuits.com/blog/articles/h-bridge-secrets/lock-anti-phase-drive/. [Dostopano 25. 1. 2017].

[4] M. Ambrožič. *Zvonarstvo na Slovenskem*. Acta ecclesiastica Sloveniae, 1993.

[5] G. Bervar. *C++ na kolenih*. Študentska založba, 2008.

[6] Microsoft Corporation. Getting Started with C++ in Visual Studio. [Online]. Dosegljivo: https://msdn.microsoft.com/en-us/library/jj620919.aspx. [Dostopano 25. 1. 2017].

[7] Eran Duchan. The Dot Factory: An LCD Font and Image Generator. [Online]. Dosegljivo: http://www.eran.io/the-dot-factory-an-lcd-font-and-image-generator/. [Dostopano 25. 1. 2017].

[8] Thijs Elenbaas. Extended EEPROM library for Arduino. [Online]. Dosegljivo: http://thijs.elenbaas.net/2012/07/extended-eeprom-library-for-arduino/. [Dostopano 25. 1. 2017].

[9] Jack Ganssle. A Guide to Debouncing, or, How to Debounce a Contact in Two Easy Pages. [Online]. Dosegljivo: http://www.ganssle.com/debouncing.htm. [Dostopano 25. 1. 2017].

[10] Francis Glassborow. *C++ od začetka: uvod za programerje.* Založba Pasadena, d.o.o., 2007.

[11] M. Kovačič. *Pa se sliš ... Pritrkavanje v slovenskem in evropskem prostoru.* Zbirka Folkloristika. Založba ZRC, ZRC SAZU, 2012.

[12] Oli Kraus. M2TKLIB - A graphics user interface library for embedded systems. [Online]. Dosegljivo: https://github.com/olikraus/m2tklib/wiki/m2tklib. [Dostopano 25. 1. 2017].

[13] Macetech. ChronoDot. [Online]. Dosegljivo: http://docs.macetech.com/doku.php/chronodot_v2.0#datasheet. [Dostopano 25. 1. 2017].

[14] Tilen Majerle. Library 09- I2C for STM32F4. [Online]. Dosegljivo: https://stm32f4-discovery.net/2014/05/library-09-i2c-for-stm32f4xx/. [Dostopano 25. 1. 2017].

[15] Tilen Majerle. Library 61- SSD1306 OLED I2C LCD for STM32F4xx. [Online]. Dosegljivo: https://stm32f4-discovery.net/2015/05/library-61-ssd1306-oled-i2c-lcd-for-stm32f4xx/. [Dostopano 25. 1. 2017].

[16] Stephanie Maks. Chronodot Library for the Arduino IDE. [Online]. Dosegljivo: https://github.com/Stephanie-Maks/Arduino-Chronodot. [Dostopano 25. 1. 2017].

[17] Microchip. 24AA512/24LC512/24FC512 512k I2C Serial EEPROM. [Online]. Dosegljivo: http://ww1.microchip.com/downloads/en/DeviceDoc/21754M.pdf. [Dostopano 25. 1. 2017].

[18] ST Microelectronics. STM32F407VG. [Online]. Dosegljivo: http://www.st.com/en/microcontrollers/stm32f407vg.html. [Dostopano 25. 1. 2017].

[19] KRN sistemi. Pogon zvonov. [Online]. Dosegljivo: http://www.krn-sistemi.si/pogon-zvonov.html. [Dostopano 25. 1. 2017].

[20] NXP Ssemiconductors. UM10204 I2C-bus specification and user manual. [Online]. Dosegljivo: http://cache.nxp.com/documents/user_manual/UM10204.pdf. [Dostopano 25. 1. 2017].

[21] Solomon Systech. SSD1306. [Online]. Dosegljivo: https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf. [Dostopano 25. 1. 2017].

[22] IAR Systems. IAR Embedded Workbench user guide. [Online]. Dosegljivo: http://supp.iar.com/FilesPublic/UPDINFO/004916/arm/doc/EWARM_UserGuide.ENU.pdf. [Dostopano 25. 1. 2017].

[23] Ajay Vijayvargiya. An Idiot's Guide to C++ Templates. [Online]. Dosegljivo: https://www.codeproject.com/Articles/257589/An-Idiots-Guide-to-Cplusplus-Templates-Part. [Dostopano 25. 1. 2017].

[24] Sir Volta. I2C DMA SSD1306 128x64 OLED. [Online]. Dosegljivo: https://github.com/SirVolta/STM32-projects/tree/master/STM32F103C8/i2c_oled_new. [Dostopano 25. 1. 2017].